

ViSiCAST Deliverable D2-1: Internet Browser Plug-in

Project Number:	IST-1999-10500
Project Title:	ViSiCAST Virtual Signing: Capture, Animation, Storage and Transmission
Deliverable Type:	RP

Deliverable Number:	D2-1
Contractual Date of Delivery:	December 2000
Actual Date of Delivery:	February 2001
Title of Deliverable:	Internet Browser Plug-in
Work-Package contributing to the Deliverable:	Workpackage 2 (Multimedia and WWW Applications)
Nature of the Deliverable:	Prototype (Software Installation plus Documentation)
Author(s):	K.J. Parsons (UEA), M. Verlinden (IvD), F. Pezeshkpour (Televirtual), R. Elliott (UEA).

Abstract:

A prototype Internet Browser Plug-in has been developed with the objective of bringing virtual human signing into the WWW environment. This will provide a basis for the provision of signing services for Deaf users of the WWW. The Plug-in comprises of two ActiveX controls: the first is the Visia Avatar produced by Televirtual, the second is a wrapper for the Avatar, known as the UEAVisia3Wrapper, produced by UEA, which adds media-player type functionality to the Avatar. Example web pages have been produced that demonstrate the Plug-in providing a signing service as part of the web page content. A Weather Forecast service, designed by IvD, was chosen as a demonstrator application, and is the source of the examples.

Keyword List: Internet, signing, deaf users, WWW, ActiveX, avatar, web page

Executive Summary

A prototype Internet Browser Plug-in has been developed to implement the playing of a sequence of motion capture files. Data, in the form of XML (prototype SiGML), is passed to the Plug-in, from which it constructs a play-list of the motion capture files that require playing.

Prototype SiGML is that defined by milestone M5-10 as part of Work-package 5 in the project plan.

A description of the design and implementation of the Plug-in, which can be described as a technology deliverable, follows.

The Plug-in consists of two ActiveX controls. The first is the Avatar ActiveX control produced by Televirtual¹. The second is the wrapper ActiveX control produced at UEA², which is known as the UEAVisia3Wrapper. The UEAVisia3Wrapper adds media-player type functionality to the Avatar, by implementing Play, Pause and Stop buttons, and a Progress bar. The UEAVisia3Wrapper is able to process XML in the form of prototype SiGML, which it uses to create a play-list of motion files. Currently only Dutch SLN motion files are supported, though British BSL motion files and German DGS motion files have been captured.

The UEAVisia3Wrapper also implements functionality allowing a user to change the position and angle of the user's viewpoint. This differentiates Avatar technology from that of video in that the user's viewpoint in the latter cannot be changed.

For this deliverable, six examples of deliberately simple and minimal HTML web pages have been created that demonstrate the Plug-in. The problem domain of the examples is Weather Forecast reporting. This was chosen as it was thought to be of general interest to all of the deaf community.

A tool for the automatic creation of Weather Forecast web pages has been produced, though this is not a part of this deliverable.

A setup program has been written that installs the UEAVisia3Wrapper ActiveX control, the example web pages and the Dutch SLN motion files that are used by the examples.

A later milestone will use the technology of this deliverable and the tools developed in conjunction with it to provide a demonstrator in the form of a live Weather Forecast web page service.

Deliverable Content

The deliverable consists of a software installation, together with associated documentation. The software installation provides:

- two ActiveX controls, which enable the signing avatar to be deployed with a simple graphical user interface, in a web browser, and also in other application environments;
- a set of motion-captured deaf sign files, supporting the Weather Forecast service, and which the signing avatar can use to play sign sequences in Dutch Sign Language (SLN/NGT);
- a set of simple examples of the use of the signing avatar controls, in Weather Forecast HTML pages to be displayed in a web browser, and also in Microsoft's Visual Basic and Excel application environments.

The documents relating to this software and forming part of the deliverable are specified in the References section below. The first of these documents provides a guide to the significance of the remaining documents, in addition to describing the software installation in greater detail.

References

1. Overview of ViSiCAST Deliverable D2-1
 2. Televirtual Avatar Software Documentation - Motion Capture, Playback and Synthesis
 3. ViSiCAST UEAVisia3Wrapper ActiveX Control
 4. UEA Visia3 Wrapper Source Code
 5. ViSiCAST Weather Forecast Web Page
-
- A. Specification webpage – Requirements document for WF WWW page
 - B. Model for weather forecasts with mapping – Description of language model for WF, and mapping to sign sequences
 - C. List of Words – list of words and phrases for the WF model
 - D. ViSiCAST Weather Forecast Creator User Manual
 - E. Initial SiGML Definition (milestone M5-10)
 - F. Streaming Prototype SiGML Demonstration (Milestone M2-1)

(1) Overview of ViSiCAST Deliverable D2-1

Ralph Elliott <mailto:re@sys.uea.ac.uk>

Introduction

D2-1 (“Avatar plugin”) is essentially an enabling technology deliverable, whose central objective is to allow the project’s signing avatar technology to be deployed in a web browser. As such it represents the essential technology development step on the path from the technology available at the start of the project to the deployment of a signing avatar in a live WWW-based Weather Forecast service, which is the objective of Milestone M2-2. This milestone has been introduced into the project since the original plan was written, and is scheduled for completion in Month 20, as explained below.

The deliverable thus consists of a software installation, together with associated documentation. The software installation provides:

- two ActiveX controls, which enable the signing avatar to be deployed with a simple graphical user interface, in a web browser, and also in other application environments;
- a set of motion-captured deaf sign files, supporting the Weather Forecast service, and which the signing avatar can use to play sign sequences in Dutch Sign Language (SLN/NGT);
- a set of simple examples of the use of the signing avatar controls, in Weather Forecast HTML pages to be displayed in a web browser, and also in Microsoft’s Visual Basic and Excel application environments.

To install this software, follow the instructions in the `Readme.txt` file in the `Avatar Controls Setup` folder of the software distribution.

The documents relating to this software and forming part of the deliverable are specified in the References section at the end of this document.

Background

The main technology resource available to ViSiCAST at the start of the project was the signing avatar technology developed primarily by Televirtual for use in the predecessor projects (Simon-the-Signer and Tessa). Both these projects used motion-capture based avatar technology, in which motion-capture sessions, followed by post-processing of the recorded data, generated a lexicon of recorded “signs”, which could then be presented in sequence by the signing avatar, as specified by a “play-list”. The avatar itself was at that time available in the form of a stand-alone application, *IHost*, capable of receiving its play-list as a stream of motion data filenames delivered over a socket-based network connection. This play-list was delivered by a separate a process whose host computer could be local to or remote from that running the *IHost* application process.

A great strength of this approach is the high level of authenticity of avatar motion, due to its basis in motion recorded from a real human signer. A limitation of the approach is its comparative inflexibility: a text can only be signed if all the words or phrases it contains are available in a lexicon of motion-captured sign data. As the motion capture and the subsequent post-processing required to generate a lexicon are comparatively resource-intensive, this condition is not one that can trivially be met. Hence a significant part of the planned activity of ViSiCAST is the development of language processing techniques to support a semi-automated system for translation from text to a signing gesture notation, together with the development of techniques for driving the signing avatar, by more or less synthetic means, from a specification using this notation.

The signing gesture notation was referred to as GML in the original project plan, but is now known as SiGML – Signing Gesture Markup Language. The gestural model used in SiGML is based on that of the well-established HamNoSys notation for sign languages; however, to facilitate its handling in networked computing environments, SiGML is represented textually in the form of XML, as defined by the SiGML dtd (document type definition). A benefit of using this extensible definition framework is that SiGML can easily be extended allow signing to be defined at other levels, either in addition to the gestural level, or as

alternatives to it. In particular, a sign may be defined at the “gloss” level, or indeed by reference to a previously captured motion data file, as long as one is available for the sign in question. The initial version of SiGML, essentially a syntactically streamlined version of the “manual” component of HamNoSys is described in [E]. The extensions to the dtd needed to allow SiGML to specify motion data files is described in [F].

Avatar Plugin Software

The present deliverable provides a new version of the avatar technology, which allows the avatar to be hosted on a web page and in other application environments. The software consists of a pair of ActiveX controls.

The main one of these is the Visicast ActiveX Control [2], which contains a new version of the Televirtual avatar. Its most significant feature is that for the first time this technology is packaged not as a complete application like IHost, but as an ActiveX control, exposing APIs to the software developer which allow application software to be built around it in a flexible manner. In addition, this version of the avatar software is enhanced in several other ways:

- It uses a more realistic, higher-definition, virtual human model, developed specifically for ViSiCAST – Visia.
- It allows the programmer to specify and control one or more virtual light sources in the avatar’s environment, do define the position of the virtual camera through which the avatar is viewed, and indeed to allow the end user to control these parameters.
- Its performance, both in terms of frame-rate and smoothness of motion, is improved.
- It uses the platform-independent OpenGL graphics interface rather than Microsoft’s Windows-specific DirectX interface: this is a necessary, although not in itself sufficient, condition for the avatar software to become available on non-Windows platforms.
- In addition to programming interfaces to support the playing of motion data files, it also exposes an interface giving the programmer direct access to the boneset configuration which drives the avatar’s rendering software: this is a significant step towards supporting avatar animation driven by a SiGML (gestural) specification.

The other, subordinate, control is the UEA Visia Wrapper Control [3,4,5]. This enhances the basic functionality of Televirtual’s Visia control in two ways:

- It provides the basic Visia control with a simple GUI, in the form of a simple “Media Player”-style set of graphic controls.
- It provides an interface allowing the programmer, including the HTML/JavaScript programmer, to specify the “play-list” of motion data files which drive the avatar in the form of a SiGML document.

In combination, these two controls thus meet the essential software requirements for this deliverable: they make the signing avatar available in the form of a browser plugin:

- which can be embedded in a web page (and, incidentally, in other application environments),
- which allows the user to control the avatar, and
- which allows the programmer to drive the avatar via a SiGML play-list.

[The D2-1 description in the original plan appears to have been drafted on the assumption that animation of the avatar driven by a gestural form of SiGML would be feasible, that is, that a “GML signing tool” would be available, before completion of the deliverable. In fact, as was recognised at the initial Consortium meeting, this is quite contrary to the remainder of the plan, which specifies that the SiGML definition (D5-2) will not be available until month 15, and that a “Notation-avatar software driver” will not be available until month 24. Thus, throughout the lifetime of the project the D2-1 description has been interpreted as requiring that the avatar be driven at this stage by a form of SiGML specifying a play-list of motion data files.]

Examples

In order to provide a focus for work on this deliverable it was decided at the outset to develop a demonstrator application which would use the avatar plugin technology. It was agreed, following a proposal from IvD, that this should take the form of a Netherlands Weather Forecast web-page, to be updated daily. Supporting this has required:

- the development of a simple natural language model for weather forecasts;
- the definition of a mapping from this model to a corresponding SLN/NGT sign language model;
- the preparation (capture and postprocessing) of a lexicon of motion data files for the signs required by the latter model.

In keeping with the project's commitment to the support of multiple European sign languages, it was decided to attempt in addition to replicate the last two steps above for the other sign languages handled by the project's language processing workpackage, that is, for the German and British sign languages, DGS and BSL.

Although this demonstrator system and its supporting software are not strictly part of the present deliverable (their completion marks Milestone M2-2, due for completion in month 20), the two have in fact been developed in parallel. Thus this deliverable is in a position to include an initial version of the SLN/NGT Weather Forecast Lexicon, together with some sample weather forecast web pages which use that lexicon to present signed weather forecasts. (The capture sessions have taken place for the other two sign languages; for BSL the post-processing is close to completion, while for DGS the post-processing is somewhat further from completion.)

The Weather Forecast language model and mapping are described in [B]; the corresponding word/phrase list is described in [C]. The current version of the Weather Forecast web page creator software is described in [D]. The web page itself is described in [5].

In addition, the deliverable includes the standard examples supplied with the Televirtual Visia Avatar Control, a simple web page, a Visual Basic application, and an Excel application, which latter two both illustrate how the control can be used to manipulate the avatar's underlying boneset. All these examples, as well as the Weather Forecast examples, are accessible through the Visicast Avatar Controls program group (sub-menu) provided by the installer.

References (Deliverable D2-1 Documentation)

1. Overview of ViSiCAST Deliverable D2-1 (this document)
2. Televirtual Avatar Software Documentation - Motion Capture, Playback and Synthesis
3. ViSiCAST UEAVisia3Wrapper ActiveX Control
4. UEA Visia3 Wrapper Source Code
5. ViSiCAST Weather Forecast Web Page

Appendices

- A. Specification webpage – Requirements document for WF WWW page
- B. Model for weather forecasts with mapping – Description of language model for WF, and mapping to sign sequences
- C. List of Words – list of words and phrases for the WF model
- D. ViSiCAST Weather Forecast Creator User Manual
- E. Initial SiGML Definition (milestone M5-10)
- F. Streaming Prototype SiGML Demonstration (Milestone M2-1)

(End of Document)

(2) Motion Capture, Playback and Synthesis

An Overview of Televirtual's Technology Contribution to Visicast

Farzad Pezeshkpour

Televirtual

1. Motion Capture Devices

1.1. *Ascension Flock of Birds*

This is an electromagnetic motion tracker, with each signal processing capable of tracking up to four sensors. The maximum sample rate is 144Hz. Each sensor generates six values for position and orientation. The signal processing unit interfaces to the host PC using TCP/IP.



1.2. *Virtual Technologies CyberGlove*

This device is a low profile glove with support for either eighteen or twenty-two resistive sensors. Each sensor generates a value between 1 and 255 inclusive, which is non-linearly proportional to the flexion of the joint under the given sensor. The maximum sample rate is 150Hz. The mapping of sensor value to joint angle changes with size and shape of the hand, as well as changes in temperature. The sensor information from the gloves is fed into a separate signal-processing unit and then passed on to the host computer using an RS232 interface.



1.3. *Motion Analysis Face Tracker*

This system tracks sixteen face markers simultaneously, at a maximum rate of 60Hz. The output is fed into PC running a DOS program, which removes any noise and disambiguates markers. The user is responsible for setting the names and order of the markers. The processing PC outputs its data via the RS232 serial port to the main capture computer.



2. Motion Capture, Playback and Synthesis

Televirtual's architecture for motion capture and animation is called Mask. The underlying work is continuing externally to the Visicast project and is proprietary to Televirtual.

2.1. *Mask Architecture*

The design of Mask was firmly concerned with a number of constraints.

- The architecture had to execute on a standard PC, running either Windows 95 or 98. Mask had to cope with capturing data from several high-frequency motion-capture devices. Therefore it needed to respond to real-time events on an operating system that isn't designed specifically for such a task.
- Mask had to run in conjunction with computationally intensive modules, such as graphics rendering engines and speech recognition systems.
- Mask had to be a reusable architecture and class framework in order to meet Televirtual's long term projects.

2.2. *Graph Model*

The Mask architecture is based upon the notion of a demand driven, directed acyclic graph. Each node implements one of many generic Mask concepts such as data-buffer; device-driver; auto multithreaded driver and buffer; data converter; motion blender or motion combiner. A typical graph for capturing motion data is shown in Figure 1.

These generic node types will be discussed in detail later. However it is important to note that at an early stage of design, it was decided that these node types (and their generic behaviour) were to be implemented using C++ templates. Using C++ templates gives the benefits of object-oriented abstraction of behaviour over similar types but in a run-time efficient manner. This is because C++ templates are in-lined by the compiler and as a result generates rather larger but faster executables.

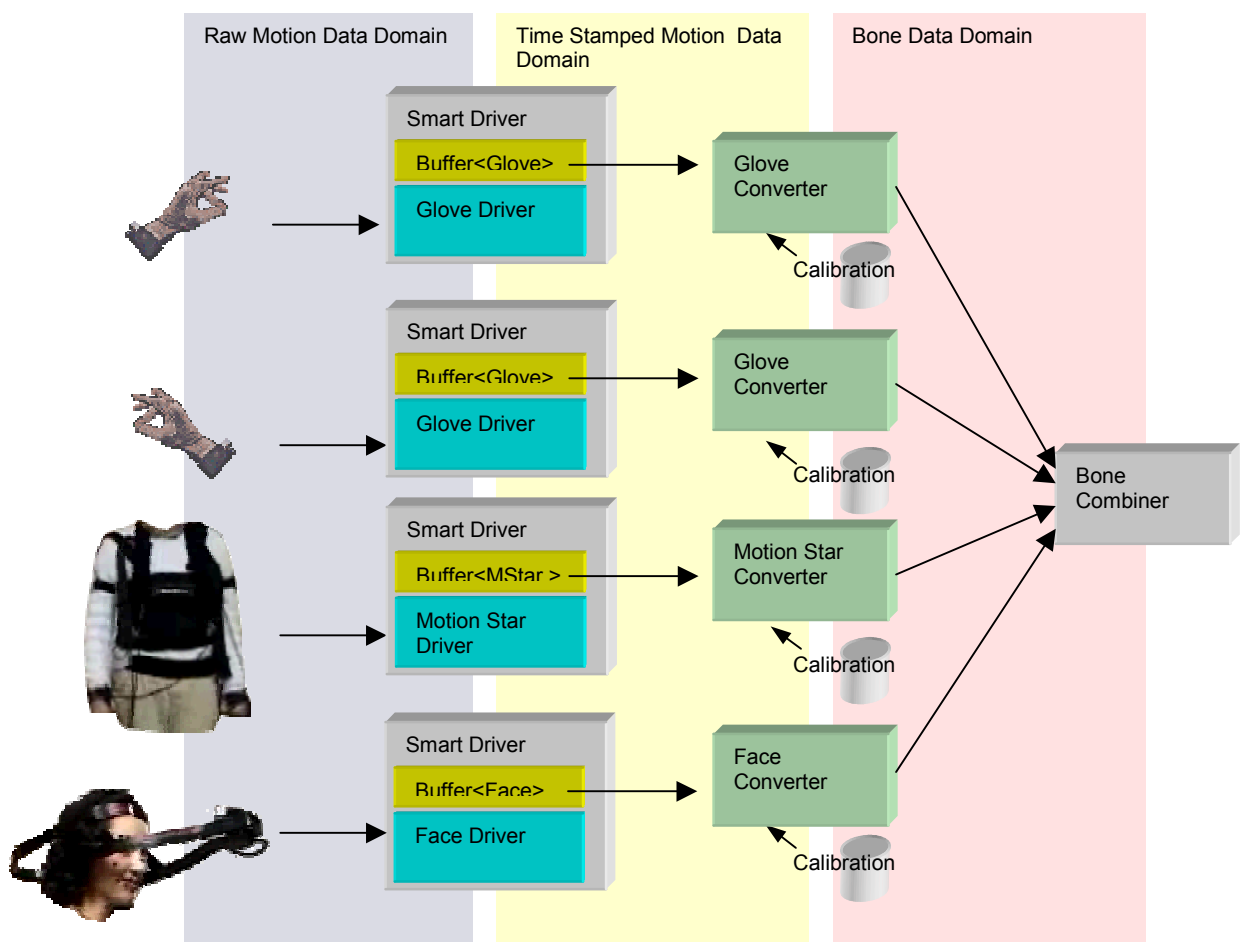


Figure 1 - A Typical Mask graph for Motion Capture.

2.3. Mask Abstract Classes

2.3.1. Pipeline

All Mask graph objects are derived from the pure virtual Pipeline template class. This template represents a given node on the Mask graph, and the collection of paths leading to that node. It defines operations to be carried out on that node, and to be propagated to all nodes on the collection of incoming paths.

The Pipeline template class abstracts the notion of a buffer of time-stamped motion data as shown in Figure 2. Motion data at this level of abstraction is either raw data from motion devices, or converted and normalised motion data. The time-stamp of each buffer element is always greater than it's predecessor, if one exists. The interval of the buffer (measured in milliseconds) can be examined, and an arbitrary start time value can be assigned to the buffer.

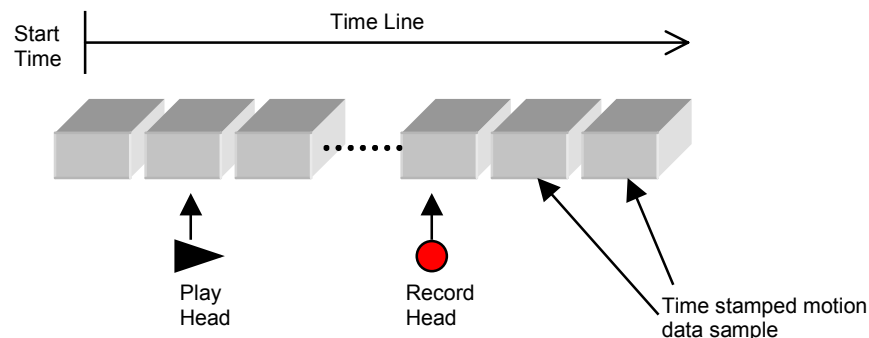


Figure 2 – Pipeline Semantics.

The data in a pipeline node can be the result of merging two or more motion data streams, each created from a different pipeline (e.g. when merging samples from disparate devices), and which may not begin at the exactly the same point in time. This is exemplified in Figure 3, where two pipelines are joined together with different

starting times. Operations on the resultant pipeline need to respect the time offset between the two source pipelines.

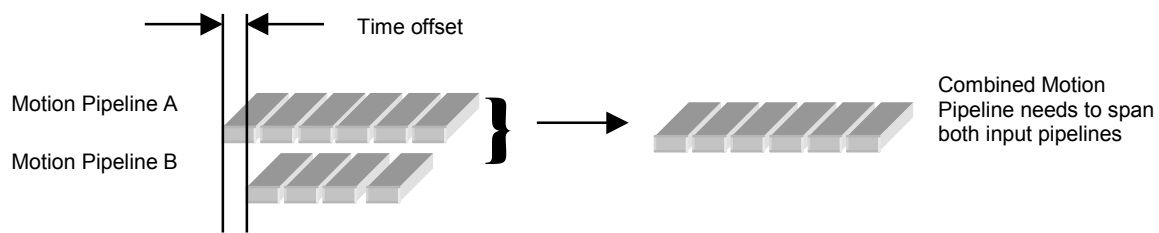


Figure 3 - A Pipeline Comprised of Two Other Pipelines that Do Not Begin at the Same Time.

For example, when setting the start time on a pipeline, it is the responsibility of the pipeline implementation to delegate the start time correctly to its input pipelines, so that all input sources maintain their relative offsets from each other.

Pipelines have the notion of a Record and a Play head (see Figure 2). The Record head indicates the location that the next item of motion data will be recorded into, and the Play head indicates the next item of motion data to be read. The play head can be positioned using either the absolute index of a buffer item or a specific point in the buffer's timeline.

To summarise, a pipeline has the following features:

- An arbitrary buffer of time-stamped, motion data that can be resized;
- A Record Head that indicates the location of the next recorded item;
- A Play Head that indicates the location of the next item to be read;
- Operations to start and stop recording of data;

- Operations to read data from the buffer; and
- Operations to save and load a buffer.

2.3.2. Smart Buffer

The SmartBuffer template class implements all buffer read-write access and persistence operations declared by the Pipeline template onto a resizable, protected, contiguous array of data. The template is parameterised on the data type of a buffer element.

2.3.3. Driver

Fundamental to Mask's capture architecture is the abstraction of a device driver. The Driver template class is a pure virtual class, and abstracts the notion of any device driver in Mask. The definition of the Driver template is such that implementation of any derived class is concerned purely with opening, closing the device, starting and stopping the capturing state of the device, and retrieve data values from the device. Most importantly, a Mask driver doesn't concern itself with the issues of multi-threading. It is expected that reading a data sample from a device will simply block the current thread until a value is returned from the device. This significantly reduces the overhead of writing drivers for new devices.

The Driver template is parameterised on two data types: the first used for initialising the driver and the second used to store raw data from the device.

2.3.4. Smart Driver

As stated in the previous section, the implementation of the low-level device drivers is not required to anticipate or implement any multithreading. In Mask, the SmartDriver template class abstracts the notion of a multithreaded device-driver. The SmartDriver class is parameterised on a data format (DataFormat), a device driver (DriverT) and a device driver initialisation structure (DriverInfo) respectively. Data transfer from a device via its device driver is managed in a secondary helper thread. This thread is created once on a lazy basis, and is activated only during capture – at other times, it is in a blocked state. The thread is given a real-time priority level in order to accommodate devices initialised to capture at a high rate. During capture, because the device driver is expected to block until new data is made available by the device, this thread priority setting is efficient.

2.3.5. Bone Data

In Mask, all motion capture data is converted into a standardised data type called BoneSet. A BoneSet is a collection of bones; the collection is analogous to the skeletal and muscular hierarchy and structure in the human body. Each bone is defined as a 3D position, a co-ordinate system defined using three axes, and a length – this format uses thirteen floating point numbers for its representation – location(3), Xaxis(3), Yaxis(3), Zaxis(3) and length. Figure 4 illustrates the definition of a bone in Mask.

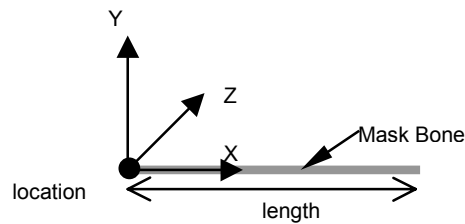


Figure 4 - Definition of a Mask Bone.

In addition to containing the collection of bones, the BoneSet has a reference to a BoneSetDesc object. The BoneSetDesc object defines the ordering, names and hierarchy of the bones within the BoneSet. The BoneSetDesc class provides functions to programmatically set-up a skeletal hierarchy, or to examine and modify an existing hierarchy.

As a result of the functionality provided by the BoneSetDesc class, the BoneSet class declares a function, *Localise*, for transforming the bone data from values in the global co-ordinate system to one where each bone is defined in terms of its parents co-ordinate system. *Localise* has an inverse function called *Globalise*, which transforms localised bones back to the global co-ordinate system.

2.3.6. BoneSource Pipeline

Mask explicitly categorises a pipeline that provides bone data called PipelineBoneSource. This category of pipeline inherits from a pure virtual class called BoneSource. BoneSource declares functions to retrieve a BoneSetDesc object that describes the hierarchy, names and ordering of bones generated by such a pipeline.

2.3.7. Converter

Mask defines a class hierarchy for converting raw motion capture data to bone data.

At the root of this hierarchy is the template class `GenConverter`. This defines the generic set of functions supported by all motion data converters.

The template parameters define the input and output types of the converter. The input of the converter is defined by the template parameters `ClassSinkT` and `DataSinkT`.

These represent the name of the class that provides the input data, and the type of input data respectively. Similarly, template parameters, `ClassSourceT` and

`DataSourceT` define the output of the converter. `ClassSourceT` represents the additional behaviour that is supported by this converter node – this can be seen in that `GenConverter` inherits from this template class parameter. `DataSourceT` is the data type that is provided at the output of the converter.

`GenConverter` declares one pure virtual function, *Convert*, that takes as its parameters a pointer to an input data element, and a pointer to an output data element. This is declared as follows:

```
virtual BOOL Convert( DataSinkT* raw , DataSourceT* converted ) = 0;
```

It is the responsibility of a derived class to implement this for the conversion from the input type to the output type.

The next level of inheritance, the `Converter` template class, specialises the `GenConverter` template class, by indicating that the input and output class types are both Pipeline objects also.

The final level of specialisation in the converter hierarchy declares the classes for the conversion of motion capture data from specific devices.

It is important to note that the conversion of raw face data to bones is a two-stage process. The first stage maps the face points of the human signer to equivalent points on a 2D plane parallel to the face of the avatar. This mapping is specified at the calibration stage. The second converter stage projects the points on the plane facing the avatar's face on to contour map of the avatars face, thereby generating the required 3D points. This projection's results are acceptable and the method is very fast. The necessity to break this process into two explicit stages was because blending between 2D points generates better face shapes than blending 3D points.

2.3.8. Combiner

Once the data from each type of motion buffer has been converted into BoneSets, it is necessary to combine them into one single BoneSet. The BoneCombiner class implements a PipeLineBoneSource, emitting BoneSet data. It has weak references to zero or more PipeLineBoneSource objects (templated also on BoneSet data), which it uses to combine into its output data.

In addition to implementing standard Pipeline functions, BoneCombiner exports functionality to add and remove input pipelines, and to define the resultant hierarchy of the bones using a BoneSetDesc object.

The functions for adding and removing inputs (Mask terminology refers to these as data sources points), the following methods have been declared in BoneCombiner:

```
// add a input source, and optionally parent root bone to  
// 'parent_bone'
```

```

void AddSource( PipelineBoneSource<BoneSet>* input,
               char* parent_bone = NULL ) ;

// As above, but also applies a transformation to the boneset prior
// to combining
void AddSource( PipelineBoneSource<BoneSet>* input, D3DMATRIX mat,
               char* parent_bone = NULL ) ;

// As above, but applies 3 rotations in the order rx, ry and rz
// instead of transformation
void AddSource( PipelineBoneSource<BoneSet>* input, double rx,
               double ry, double rz, char* parent_bone = NULL ) ;

// As above, but follow rotation with a translation
void AddSource( PipelineBoneSource<BoneSet>* input, double rx,
               double ry, double rz, double tx, double ty,
               double tz, char* parent_bone = NULL ) ;

// Removes an input source
void DeleteSource (PipelineBoneSource<BoneSet> * input);

```

2.3.9. Blender

In order to blend smoothly between motion data from one stream to the next, Mask defines the Blender class hierarchy.

The blender semantics in Mask are as follows: each blender takes a number of input sources. One source is declared as the ‘background’ source, whilst the others are declared as ‘foreground’ sources. The blender allows the background source through only if there aren’t any unplayed foreground moves to display. This architectural design allows for the avatar to have an ambient motion whilst it hasn’t been requested to display any ‘foreground’ motion.

Each foreground source can be added to the blender parameterised by a relative weight and one of either two blending functions: *Average* or *Replace*. The weight describes an importance in the blending function for that motion source.

Average blending is the typical form of blending where data from one motion capture pipeline is blended smoothly into another motion capture pipeline, as a function of time.

A motion pipeline that has been assigned for *Replace* blending overrides all other motion data when it becomes active. An example of *Replace* blending is when the avatar face bones are being driven by phonemes extracted from a speech recognition system. In this case the mouth shape motion-stream overrides any other face motion data in order to change the shape of the lips correctly.

3. Graphics

3.1. Mesh Animation

Animation of a mesh can be performed using three different techniques: segmented mesh, mesh deformation and morph targets.

The segmented mesh technique divides the mesh into distinct and separate segments at the location of skeletal joints. Typically the segments are joined together in a hierarchy. Animation of a mesh segment is achieved by modifying its local transformation matrix.

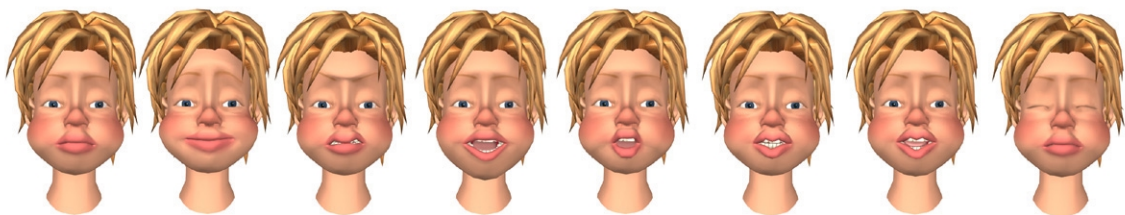


Figure 5 - A series of facial morph targets. A morphing system is capable of blending one or more of these face targets together to synthesise a new face shape.

Conversely to the segmented mesh approach, mesh deformation (commonly referred to as ‘smooth skinning’ by graphics artists) uses a seamless mesh. Each vertex of the mesh is ‘attached’ to one or more skeletal bones. An attachment is given in terms of 3D point in the local co-ordinate space of the bone, together with a weight. To deform the mesh, each vertex is visited in turn. The global location of the vertex is computed as a weighed average of its attachments.

A variation on the mesh deformation technique is morph targets. This technique works best with simple objects without multiple joint linkages. It is ideally suited for the face. Given a suitable object, a set of mesh shapes is made that define the average shape for the mesh, as well as its extremes. For examples, with a face, there will be an average, emotionless face, together perhaps with a smiling face, a frowning face and a raised eyebrow face. Each ‘target’ mesh is assigned a weight. The resultant mesh is calculated on a per vertex basis. A vertex’s position is the weighed average of that vertex in all targets. This is an effective technique for an animation system that works directly with emotional expressions and phonetic mouth shapes.

Televirtual’s animation architecture implements both mesh deformation and morph targets. Up till now, the Visicast project has required the mesh deformation technique for it’s animation – however, it is envisaged that morph targets may become useful in the synthesis of face shapes.

3.2. *Rendering of the Avatar*

The rendering of the Avatar is performed by a component referred to as the COMRenderer. This component abstracts the notion a 3D scene and the rendering of the given scene within a graphics window. A COMRenderer scene is has lights

(ambient, directional, positional and spot lights), a camera, and zero or more 3D models. The COMRenderer exposes its functionality as a set of COM types (interfaces, structures, enumerations etc).

Previous rendering engines developed by Televirtual where strongly tied to the Microsoft DirectX API. In order to improve portability of the rendering code, the COMRenderer is implemented using OpenGL.

4. Calibration

The conversion from raw device data to bone data requires a calibration of the sensors with respect to the human and the virtual signer. Each motion data source has for this purpose a calibration editor.

4.1.1. CyberGlove Calibration Data

Calibration data for the hands is a piece-wise non-linear function that maps the values from the sensors onto angles. This function is graphically achieved whilst the performer wears the data glove, and executes a series of predetermined positions. For each position, one can select any joint in the hand to display its mapping function graphically and to adjust the curve using a mouse (see Figure 6).

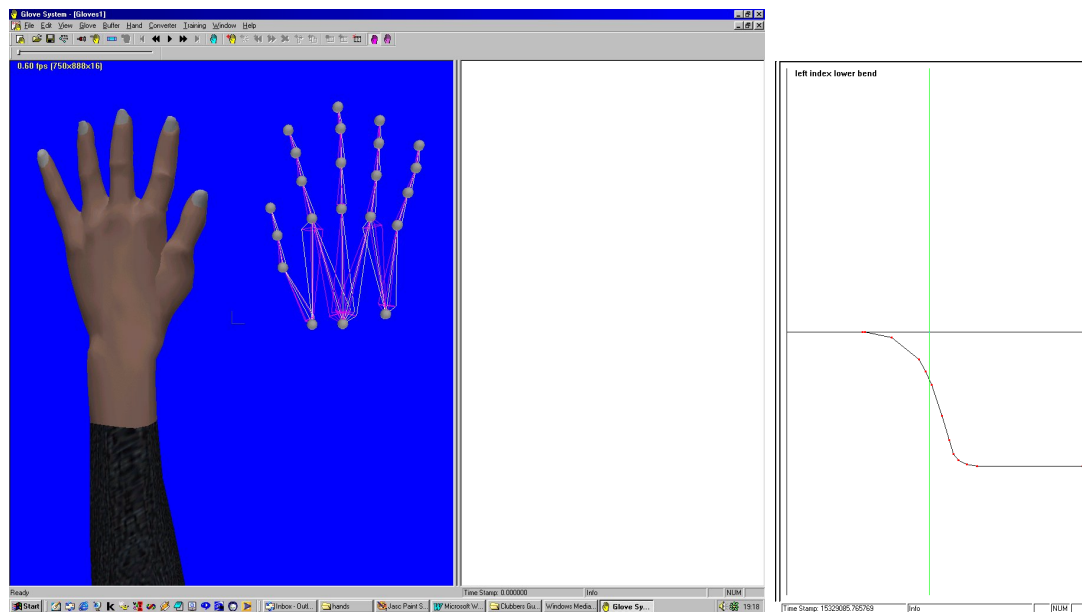


Figure 6 - Screenshot of the Glove Editor.

The glove data converter uses the calibration data to firstly generate a set of rotations for each degree of freedom in every joint. It then applies these rotations to the default calibration hand.

4.1.2. Face Tracker Calibration Data

Calibration data for the face is also a two-stage process. The first stage maps the markers to a plane in front of the avatar's face, such that the virtual markers are in the correct location (Figure 7(i)). The second stage isn't really a calibration, but a transformation from the 2D virtual markers to the 3D face. This is achieved by defining for each avatar a 3D height-field that the 2D virtual markers can be projected upon. From these project 3D markers, the bones of the face can be constructed (Figure 7(ii)).



Figure 7 - (i) Mapped markers shown in red are projected to underlying cast to create the bone locations. (ii) Face mesh warped to the face bones.

4.1.3. Motion Star Calibration Data

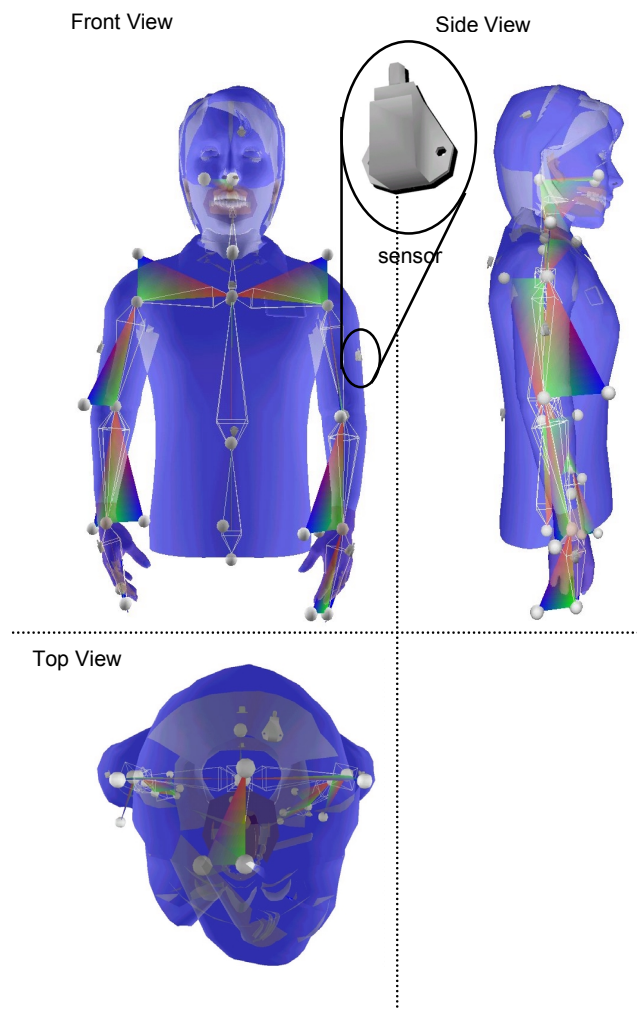


Figure 8 - Positioning of the Motion Star Sensors on the Avatar Body.

The calibration data for the Motion Star defines the location of each sensor with its respective bones. Sensors are attached to the upper arms, hands, upper back, lower back, and head. The Motion Star converter from these markers constructs the stance of the entire body. In order to calibrate the converter, the performer stands in what is deemed to be a standard pose. Within the Motion Star calibration application, the locations of the bones that have had a sensor attached are moved into a position that matches the stance of the performer

as shown in Figure 8. This informs the converter of the location of each sensor from its respective bone. The performer then begins to carry out a few more standard stances in order to check if the calibration is correct. At this stage minor modifications to the stance may need to be made.

5. Motion Files

Mask stores its motion files using COM's compound storage API. This API defines a file system within this file. Figure 9 shows a motion file being viewed in a Mask application called Storage, which visualises the contents of the motion file as a tree-view. Within a motion file there are four folders – one for each of the motion sources – face, gloves and body – there exists a folder that contains two 'files' containing the raw motion data and the calibration data required in converting the raw data to bone data. In addition there exists two files containing the blending regions for the motion.

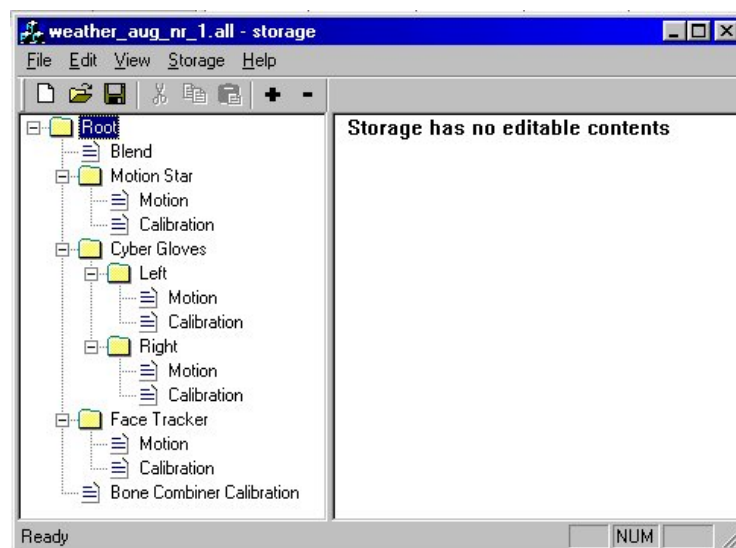


Figure 9 - Structure of a Mask Motion File.

The calibration data is a relative path reference to the device specific calibration file.

6. Visicast ActiveX Control Architecture

6.1. Overview

The animation and rendering systems from Televirtual can easily be adapted for different types of plug-in or component architectures. On the Windows platform, the most common method of componentisation is using the Component Object Model. It is not within the scope of this work to describe this component architecture, but it will suffice to say that COM enables the location-transparent invocation of function groupings called interfaces across thread, process, machine boundaries with variable degree of bindings to different languages.

COM defines sets of system-wide interfaces that handle such notions as lifetime management, persistence and embedding of windows. These interfaces are given the umbrella term ActiveX. One important definition within ActiveX is in-place activation: this is a protocol between a container application and a component that exhibits a user interface. The protocol defines how the container and component can co-operate together to create a seamless user interface (such as sharing of menus, toolbars etc).

6.2. Sub Components and Threads

The Visicast ActiveX component hosts four instances of other components:

- IHostCOM – a componentised version of the Mask animation system. This exposes a series of interfaces that enable the queuing of motion capture moves for playback, and the acquisition of bone data.

- FastMeshCOM – an optimised module used for mesh deformation and morph targets.
- COMRender – the rendering component that is described in section 3.2.
- MSXML – The Microsoft XML engine, used in reading boneset hierarchy data.

The Visicast control creates a single helper thread that is responsible for periodic acquisition of animation bone data from IHostCOM, the warping of the mesh and the rendering of the avatar. This is illustrated in the following Figure 10. IHostCOM is called by the helper thread to retrieve the bone data for the current frame. Next, FastMeshCOM is called to deform the avatar model stored within the 3D scene. Finally the COMRender is called to render the scene.

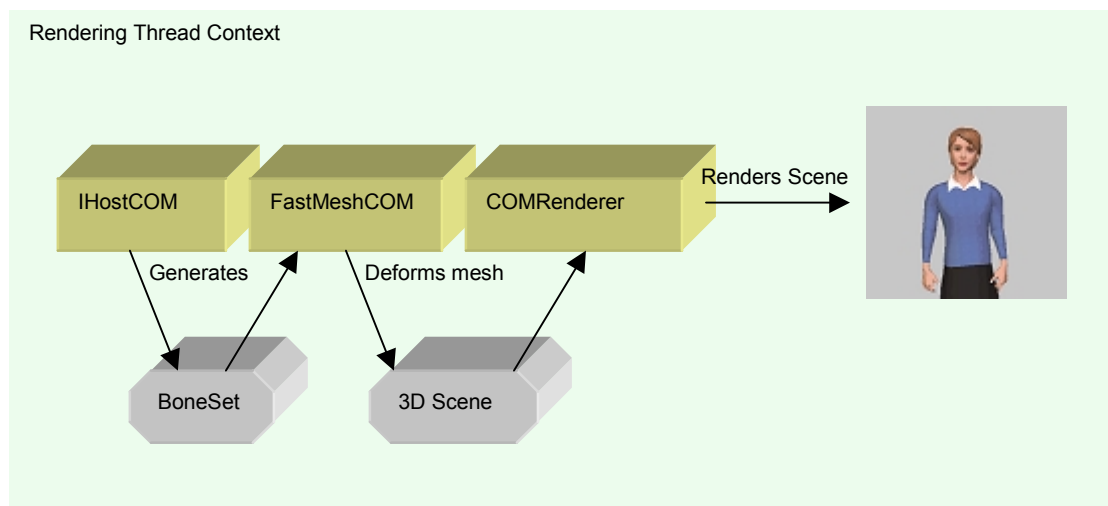


Figure 10 - Operations carried out within the context of the rendering thread.

Appendix A. The Visicast ActiveX Control Interfaces

IVisicastControl Interface

IVisicastControl is the primary interface for programming the avatar.

Properties

Property	Type	Description
BackColor	OLE Color	Colour of the background. Defaults to black. Read-Write property.
Avatar	string	The name of the current avatar. Read-Write property. Setting this to a different value causes control to load the new avatar. Avatars are stored under the 'avatars' subdirectory within the installation.
Camera	ICamera	The Camera object for this avatar.
Lights	ILights	The collection of lights used in the scene.
DirectControl	bool	Read-Write property. If true, then avatar ignores animation stream, and allows for direct manipulation of posture, face and hands.
BoneSet	IBoneSet	Set of bones for the avatar. Changing these, whilst DirectControl is true, alters the shape of the avatar.

Methods

Method	Description
PlayAnimation (string animation, float speed = 100%)	Requests to queue an animation file for playback at some factor given by <i>speed</i> as a percentage of the original animation speed. <i>Speed</i> is optional and defaults to 100%.
StopAll ()	Called to stop all animation and clear the animation queue.
DoAmbient ()	Performs an ambient move.

_IVisicastControlEvents Event Interface

The control exposes a set of events through the _IVisicastControlEvents interface.

Events

Event	Description
OnInitialised ()	Called when control is initialised.
OnDecreaseQueueLength (long items)	Called when a pending animation is moved off the queue to be played. Passes the current length of the queue in parameter <i>items</i> .
MouseDown (bool bShift, bool bControl, short button, long x, long y)	Mouse down event. bShift and bControl give the state of the shift and control keys. Button is combination of the following flags: 1:=Button 1; 2:=Button 2; 4:=Button 3.
MouseUp (bool bShift, bool bControl, short button, long x, long y)	As above, but for mouse up event.
MouseMove (bool bShift, bool bControl, short button, long x, long y)	As above but for mouse move event.
MouseWheel (bool bShift, bool bControl, short button, long x, long y)	As above but for mouse wheel event.

IBoneSet Interface

The IBoneSet interface provides read and write access to the avatars bone data as described section 2.3.5. The features provided by this interface are not currently available for pure scripting languages such as JScript, VBScript or PerlScript. A future release of this software may address this issue with additional and more granular functions.

Properties

Property	Type	Description
BoneCount	long	Read only count of bones in the boneset.
BoneSetDesc	IBoneSetDesc	Access to the description of the bone set's hierarchy.

Methods

Method	Description
SetBones (bool bLocal, SAFEARRAY(float) ** bones)	Sets the state of the bones using a SAFEARRAY of floats. Bones can be set either in local or global state according to the value of the bLocal parameter. If true, the BoneSet will expect a localised set of values.
GetBones (bool bLocal, SAFEARRAY(float) ** bones)	Reads the set of bones into a SAFEARRAY of bones that must be allocated by the caller. Bones will be either localised or not according to the value of the bLocal parameter.
SetBone (long boneIndex, bool bLocal, SAFEARRAY(float) ** bonedata)	Sets the values for a single bone. BLocal defines whether bone is localised or not.
GetBone (long boneIndex, bool bLocal, SAFEARRAY(float) ** bonedata)	Retrieves the values for a single bone in a SAFEARRAY of floats, pre-allocated by the caller. bLocal determines if the bone is localised or not.

IBoneSetDesc Interface

The IBoneSetDesc interface gives a representation of the avatars hierarchy, which can be both read and modified.

Properties

Property	Type	Description
BoneCount	long	Read only count of bones in the bone set description.

Methods

Method	Description
Create (long cBone)	Clears any previous description. Creates the space for a bone set description of <i>cBone</i> bones.
GetParent (long iBone, long * piParent)	Retrieves the parent of a bone.
GetChildCount (long iBone, long * piChildCount)	Retrieves the number of children held by a bone.
GetName (long iBone, string * pName)	Retrieves the name for a bone.
SetName (long iBone, string name)	Sets the name for a bone.

SetParent (long child, long parent)	Sets the parent for a bone.
GetBoneIndex (string name, long * piBone)	Retrieves the index for a bone given by name.

ICamera Interface

The ICamera interface allows for a programmer to dynamically change the position, orientation and ancillary settings of the control's camera. For some reason, I found myself creating properties starting with lowercase letters ... apologies for the inconsistency.

Properties

Property	Type	Description
location	IVertexV	Location of the camera.
face	IVector	Facing direction of the camera.
up	IVector	Defines the roll of the camera, by giving the camera's up vector.
fov	float	Field of View. Read/Write.
aspect	float	Aspect ratio of the camera. Read/Write.
nearZ	float	Near clipping Z value. Read/Write.
farZ	float	Far clipping Z value. Read/Write.

ILights Interface

This is collection object allowing for the enumeration, creation and manipulation of lights within the scene.

Properties

Property	Type	Description
Count	long	Count of current lights in the scene. Read only.
Item	ILight	Retrieves a specific light, given either a numeric or by-name index. The name is part of the COM standard for collection interfaces.

Methods

Method	Description
ILight * Create (string name)	Creates a new light within the scene
Delete (variant light)	Delete a light, specified either by its name, or by reference.

ILight Interface

This interface is provided by all lights within the control's scene (apart from the ambient light).

Properties

Property	Type	Description
lightType	TVR_LightType	The type of light.
ambient	IColour	The ambient colour of the light, if <i>lightType</i> is global.
diffuse	IColour	Diffuse colour of light.
specular	IColour	Specular colour of light.
cutOff	float	Read/Write.
exponent	float	Read/Write.
constAttenuation	float	Read/Write.
linearAttenuation	float	Read/Write.
quadraticAttenuation	float	Read/Write.
position	IVertexLP	Position of light.
direction	IVertexV	Direction of light.
name	string	Read/Write name of light.

IVector and IVertexV Interfaces

Both of these interfaces describe essentially a triplet of floats, but behave within different domains.

Properties

Property	Type	Description
----------	------	-------------

x	float	Read/Write.
y	float	Read/Write.
z	float	Read/Write.

IVertexLP Interface

This interface is used to describe a weighed interface

Properties

Property	Type	Description
x	float	Read/Write.
y	float	Read/Write.
z	float	Read/Write.
w	float	Read/Write

IColour Interface

Used to manipulate colour values. The value of each property must be in the range 0.0 - 1.0.

Properties

Property	Type	Description
r	float	Read/Write. Red component.
g	float	Read/Write. Green component.
b	float	Read/Write. Blue component.
a	float	Read/Write. Alpha (transparency) value.

TVR_LightType Enumeration

TVR_LightType is an enumeration that declares the following values: TVR_DIRECTIONAL (0) TVR_POSITIONAL (1) TVR_SPOT (2) TVR_GLOBAL (3)

(3) ViSiCAST UEAVisia3Wrapper ActiveX Control

1 Introduction

Deliverable D2-1 of ViSiCAST Work Package 2, is a prototype of an Internet Browser Plug-in, which provides signing by an Avatar that is part of a web page. The Plug-in processes SiGML and provides signing by an Avatar that is contained within a web page. The Plug-in consists of two ActiveX controls, the Televirtual Avatar[4] and the UEAVisia3Wrapper. This document describes the UEAVisia3Wrapper.

2 Overview

An ActiveX control, called UEAVisia3Wrapper, was developed implementing Play, Pause and Stop buttons, and a progress bar. This was implemented to add a RealPlayer media-player type user interface to the Televirtual Avatar ActiveX control as well as controlling the Avatar.

The 'UEAVisia3Wrapper' ActiveX control implements a play-list of motion files and these are fed to the Avatar as appropriate. There are two methods available for a user of the 'UEAVisia3Wrapper' to specify the motion files to be played. The first is to pass the wrapper control an XML document that contains SiGML defining the motion files to be played. This is implemented by the 'ReadSiGML' method of the wrapper control. The second method is called 'AddToPlayList'. This accepts a motion file-name as its single parameter. Each invocation of the 'AddToPlayList' method adds the file passed as a parameter to the play-list.

This design encapsulated complication inside the UEAVisia3Wrapper ActiveX control and made the construction of the Web Page simpler. It also allows the UEAVisia3Wrapper ActiveX control to be hosted in other environments that support ActiveX controls.

3 Design and Implementation

3.1 Implementation Tool

The 'UEAVisia3Wrapper' ActiveX control was built using Microsoft Visual Basic 6.

3.2 SiGML

The 'UEAVisia3Wrapper' has the ability to accept 'prototype SiGML': when given an XML document object that contains details of the motion files to be played. This is handled by the 'ReadSiGML' function.

The addition of this functionality means that Microsoft's MSXML3 Parser needs to be installed for the UEAVisia3Wrapper to work, [1]. (MSXML3 is also required by the Televirtual control.) To support this, the SiGML DTD file (and the SMIL DTD file) must be in the same folder as any HTML file that contains the 'UEAVisia3Wrapper', [2][3].

3.3 States

The UEAVisia3Wrapper implements maintains an internal finite-state machine which controls the progress of the avatar and reflects the state of its graphical user interface. The seven states identified are:

0. Setting Up
1. Ready To Start
2. Playing
3. Playing Paused
4. Ready To Resume
5. Playing Halted
6. All Played

Obviously, some transitions are not possible E.g. from Ready-To-Start to All-Played. All legal state transitions are documented in the AvatarController.ctl source code file.

For each of the seven states there are private member functions to put the control into that state, E.g. 'SetStateSettingUp', 'SetStatePlayingHalted' Etc.

3.4 Initialising the Avatar

When the Initialize event is fired the Avatar is initialised by the private function 'StartVisiaAvatarControl', which sets up the Avatar's camera, lights and avatar properties.

3.5 Avatar Events

3.5.1 Initialised

This is fired when the Avatar has finished initialising itself with the property values given to it by the 'StartVisiaAvatarControl' function and is ready to start playing signs.

3.5.2 OnDecreaseQueueLength

This event is fired by the Avatar when a file passed to it has been removed from its queue, i.e. it is about to be played. The behaviour of the code for this event is documented in the source code.

3.5.3 MouseDown and MouseMove

These events are used to provide a user interface for the easy movement of the user's viewpoint.

Positioning the mouse on the avatar display panel and dragging upwards causes the user's viewpoint to move further away from the Avatar, so the Avatar appears to recede into the distance. Conversely, dragging downwards causes the user's viewpoint to move towards the Avatar, so the Avatar appears to move closer.

A leftward drag causes the user's viewpoint to rotate to the Avatar's left, that is, the Avatar appears to turn to its right. A movement to the right causes the user's viewpoint to rotate to the Avatar's right, so the Avatar appears to turn to its left.

With the 'Ctrl' key pressed, an upward drag of the mouse causes the user's viewpoint to move downwards, that is, the Avatar apparently moves upwards. Similarly, Ctrl-Dragging downwards causes the user's viewpoint to move upwards, so the Avatar appears to move downwards.

3.6 Play-list

The play-list is implemented as an array of strings. This is initialised by the 'ReadSiGML' method using the motion file identifiers contained within the SiGML. Or it can be constructed by a sequence of calls of the 'AddToPlayList' method. During each call of this method the size of the array is increased by one, and the name of the file passed as a parameter is added to the array.

3.7 Buttons

The buttons are enable and disabled as appropriate depending upon the current state of the 'UEAVisia3Wrapper'. Clicking on the 'Play' button causes any items in the play-list to be fed to the Avatar to be played. Clicking on the 'Pause' button causes the playing of signs to be paused. Clicking on the 'Stop' button stops the playing of signs and returns the player to the start of the play-list. Clicking on the 'Play' button after a pause causes playing to be resumed at the point in the play-list where the pause occurred.

3.8 Cycle Checkbox

If the 'Cycle' check-box is checked the 'UEAVisia3Wrapper' repeatedly plays all the files in the play-list. This is a modification of the normal termination behaviour of the playing cycle. This may be removed from future releases, since it was included primarily as an aid to testing.

3.9 Feeding File Names To The Avatar

The constant value 'IFeedSize' defines how many motion files are fed to the Avatar in one go. Currently this value is set at 3. Files are fed to the Avatar in this way to allow it to cache motion files prior to playing and to carry out the blending of successive pairs of sign files.

3.10 Slider Control

The Slider control moves to indicate the progress made through the signing play-list when the Avatar is playing. The slider control is initialised by the 'UpdateSliderLimits' function and updated by the 'UpdateSliderDisplay' function.

It can also be used, when the player is idle, to indicate where to start the playing of signs, by dragging the thumb of the slider to a position along its scale and then pressing Play

References

- [1] Microsoft Corporation XML Developer Center home page: <http://msdn.microsoft.com/xml/default.asp>
- [2] Professional XML by Richard Anderson, et al, Wrox Press; ISBN: 1861003110
- [3] World Wide Web Consortium (W3C) XML home page: <http://www.w3.org/XML/>
- [4] (D2-1 Document 2) Motion Capture, Playback and Synthesis – Televirtual Avatar Documentation

(4) UEA Visia3 Wrapper Source Code

Visual Basic 6 Source Code

```
#####
'UEA Wrapper for Visia Avatar ("ViSiCAST Active X Control")
'Kevin Parsons and Ralph Elliott
'November, 2000
'
'This is the wrapper, or controller, that adds
'"pseudo-QuickTime" functionality to Televirtual's Visia
'ActiveX Control, which is embedded within it.
'
'December, 2000
'Additions to support adjustment of the avatar's camera view
'controlled by the mouse. -R.E.
'
'January-February, 2000
'Further additions:
'-Accept SiGML playlist KJP;
'-Avatar initialisation now assumes normalised mocap files,
'  so Visia starts off facing -z direction (l-h coordinates);
'-Camera view management code tidied up.
'R.E.
#####

Option Explicit
Option Base 0

' Avatar Wrapper States
Private Const sSettingUp As Long = 0 ' the "start" state
Private Const sReadyToStart As Long = 1
Private Const sPlaying As Long = 2
Private Const sPlayingPaused As Long = 3
Private Const sReadyToResume As Long = 4
Private Const sPlayingHalted As Long = 5
Private Const sAllPlayed As Long = 6
Private awState As Long
'
' Possible awState transitions are:
'
' sSettingUp -> sReadyToStart (on Visia-initialised and sign[s]
'                          available for playing)
' sReadyToStart -> sPlaying (on click-Play)
'               -> sReadyToResume (on "slider up from zero")
' sPlaying -> sPlaying (on "one more played but not all")
'           -> sAllPlayed (on "all played")
'           -> sPlayingPaused (on click-Pause)
'           -> sPlayingHalted (on click-Stop)
' sPlayingPaused -> sReadyToStart (on "avatar queue-length down to
'                          zero after click-Pause" with
'                          slider = 0)
'               -> sReadyToResume (on "avatar queue-length down to
'                          zero after click-Pause" with
'                          slider > 0)
' sPlayingHalted -> sReadyToStart (on "avatar queue-length down to
'                          zero after click-Stop")
' sReadyToResume -> sReadyToStart (on click-Stop or "slider down to
'                          zero")
'               -> sPlaying (on click-Play with slider < Max)
'               -> sAllPlayed (on click-Play with slider = Max)
' sAllPlayed -> sReadyToStart (immediate ("tau") with cycle OFF)
'               -> sPlaying (immediate ("tau") with cycle ON)
```

```

Private Const IFeedSize As Long = 3 'from 2 to 3, 2000-12-22
Private arsPlaylist() As String
Private IPlaylistSize As Long
Private ISentCount As Long
Private IPlayedCount As Long ' Shadows Slider value
Private bAvatarInitialised As Boolean

```

```

'Data for Mouse-Driven Avatar Camera Position Control

```

```

'Fixed Startup Values:

```

```

Private Const initBackground _
    As Long = &HFF708080
Private Const pi As Single = 3.14159265358979
'"z-limit" values are relative to camera's p.o.v.:
Private Const initZNear As Single = 250
Private Const initZFar As Single = 15000
Private Const initCamDist As Single = 3000
Private Const initCamY As Single = 125
'Add this to the near z-limit to stop zooming getting the camera
'inside poor Visia (except maybe for her arms when extended):
Private Const fatnessAdj As Single = 350
'Camera Rotation is relative to -z axis, i.e. the direction in
'which the normalised Visia faces. This adjustment converts it
'back to standard form, i.e. relative to +x axis
'(it IS a left-handed coordinate system!):
Private Const adjCamRot As Single = -90
'Limits for camera rotation are -limitCamRot..+limitCamRot:
Private Const limitCamRot As Single = 180

```

```

'coordinates of centre of rotation (c.o.r.) for camera
'(all fixed at initialisation in this implementation)

```

```

Private Const centreX As Single = 0
Private Const centreY As Single = 0
Private Const centreZ As Single = 0

```

```

'Variables:

```

```

'distance of camera from the c.o.r.
Private cameraDist As Single

```

```

'angle of rotation of camera around c.o.r
Private camRot As Single

```

```

'previous mouse XY (GUI coordinates) for dragging

```

```

Private lastx As Single
Private lasty As Single

```

```

Private Sub SetLight( _
    lightname As String, x As Double, y As Double, z As Double, _
    r As Double, g As Double, b As Double)
    Dim light As ILight
    Set light = Avatar.Lights.Create(lightname)
    light.lightType = TVR_DIRECTIONAL
    With light.position
        .x = x
        .y = y
        .z = z
    End With
    With light.diffuse
        .r = r
        .g = g
        .b = b
        .a = 1
    End With
End Sub

```

```

Private Sub SetCameraY(y As Single, corY As Single)

```

```

'parameters:

```

```

' global Y coordinates for camera and its

```

```

' centre of rotation (c.o.r.)
With Avatar.Camera
    .location.y = y
    .face.y = y - corY 'initially, the camera faces the c.o.r.
End With
End Sub

Private Sub UpdateCameraY(deltaY As Single)
    With Avatar.Camera.location
        .y = .y + deltaY
    End With
End Sub

Private Sub SetCameraXZ( _
    corX As Single, corZ As Single, dist As Single, theta As Single)
'corX: global X for centre of rotation (c.o.r.)
'corZ: global Z for c.o.r.
'dist: distance of camera from the c.o.r.
'theta: angle of rotation (in xz-plane) of camera about c.o.r.
'      (with -z direction as 0, currently)

    Dim thetaStd As Single
    'Factor in the adjustment from -z to +x direction:
    thetaStd = (theta + adjCamRot) Mod 360

    Dim rx As Single, rz As Single
    rx = dist * Math.Cos(pi * thetaStd / 180)
    rz = dist * Math.Sin(pi * thetaStd / 180)

    With Avatar.Camera
        With .location
            .x = corX + rx
            .z = corZ + rz
        End With
        With .face
            .x = (-rx)
            .z = (-rz)
        End With
    End With
End Sub

Private Sub StartVisiaAvatarControl()
'Initialise global data for camera position control
cameraDist = initCamDist
camRot = 0 'relative to -z direction, currently
lastx = 0
lasty = 0

Avatar.BackColor = initBackground

'Set up the camera ...

With Avatar.Camera
    'camera "up" vector: <YZ> from camera's pov:
    With .up
        .x = 0 'This one is ignored?
        .y = 1
        .z = 0
    End With
    'z-limits from camera's pov:
    .farZ = initZFar
    .nearZ = initZNear
End With

'Set Camera Position and Direction
SetCameraY initCamY, centreY
SetCameraXZ centreX, centreZ, cameraDist, camRot

'Specify lighting
'(These values are taken from the original IE Example Page,

```

```

'adjusted for normalisation, so someone at Televirtual thinks
'they are good.)
SetLight "light1", -0.15, -0.4, 0.7, 1, 0.85, 0.85
SetLight "light2", 8, 0, 0.7, 0.34, 0.35, 0.2

'Specify avatar model: Visia
Avatar.Avatar = "visia"
End Sub

Private Sub SetButtons( _
    bPlay As Boolean, bPause As Boolean, bStop As Boolean)
    btnPlay.Enabled = bPlay
    btnPause.Enabled = bPause
    btnStop.Enabled = bStop
    Slider.Enabled = (bPlay)
End Sub

Private Sub SetStateSettingUp()
    awState = sSettingUp
    bAvatarInitialised = False
    IPlaylistSize = 0
    SetButtons bPlay:=False, bPause:=False, bStop:=False
    Debug.Print "#### state = SettingUp"
End Sub

Private Sub SetStateReadyToStart()
    awState = sReadyToStart
    IPlayedCount = 0
    ISentCount = 0
    UpdateSliderDisplay
    SetButtons bPlay:=True, bPause:=False, bStop:=False
    Debug.Print "#### state = Ready"
End Sub

Private Sub SetStatePlaying()
    awState = sPlaying
    ISentCount = IPlayedCount
    SetButtons bPlay:=False, bPause:=True, bStop:=True
    Debug.Print "#### state = Playing"
End Sub

Private Sub SetStatePlayingPaused()
    awState = sPlayingPaused
    SetButtons bPlay:=False, bPause:=False, bStop:=False
    Debug.Print "#### state = PlayingPaused"
End Sub

Private Sub SetStateReadyToResume()
    awState = sReadyToResume
    SetButtons bPlay:=True, bPause:=False, bStop:=True
    Debug.Print "#### state = ReadyToResume"
End Sub

Private Sub SetStatePlayingHalted()
    awState = sPlayingHalted
    SetButtons bPlay:=False, bPause:=False, bStop:=False
    Debug.Print "#### state = PlayingHalted"
End Sub

Private Sub SetStateAllPlayed()
    awState = sAllPlayed
    IPlayedCount = 0
    ISentCount = 0
    UpdateSliderDisplay
    'Leave buttons unchanged, as this state is transitory
    Debug.Print "#### state = AllPlayed"
End Sub

Private Sub SetAllPlayedThenDoReset()
    SetStateAllPlayed

```

```

'Now do a further, automatic, state transition ("tau" transition)
If chkCycle.Value = 1 Then
    Debug.Print "CYCLE: Continue playing ..."
    StartPlaying
Else
    SetStateReadyToStart
End If
End Sub

Private Sub StartPlaying()
    SetStatePlaying
    FeedSignsToAvatar
End Sub

Private Sub CheckReadyToStart()
    If awState <> sReadyToStart Then
        If bAvatarInitialised And (0 < IPlaylistSize) Then
            SetStateReadyToStart
        End If
    End If
End Sub

Private Sub FeedSignsToAvatar()
    Do While (ISentCount < IPlayedCount + IFeedSize) And _
        (ISentCount < IPlaylistSize)
        Avatar.PlayAnimation (arsPlaylist(ISentCount))
        Debug.Print "Sent File:" & arsPlaylist(ISentCount) & _
            ", at index: " & ISentCount & _
            ", sent: " & ISentCount + 1 & _
            ", (PL len = " & IPlaylistSize & ")"
        ISentCount = ISentCount + 1
    Loop
End Sub

Private Sub UpdateSliderLimits()
    Slider.Min = 0
    Slider.Max = IPlaylistSize
End Sub

Private Sub UpdateSliderDisplay()
    Debug.Print "set slider -- played: " & IPlayedCount & _
        ", sent: " & ISentCount
    Slider.Value = IPlayedCount
End Sub

Private Sub Avatar_MouseDown( _
    ByVal bShift As Long, ByVal bControl As Long, _
    ByVal button As Integer, _
    ByVal x As Long, ByVal y As Long)
    ' Save x,y for use by MouseMove() [see below]
    lastx = x
    lasty = y
End Sub

Private Sub Avatar_MouseMove( _
    ByVal bShift As Long, ByVal bControl As Long, _
    ByVal button As Integer, _
    ByVal x As Long, ByVal y As Long)

    If button = 1 Then 'act on "drag", but not plain "move"

        camRot = camRot + lastx - x
        If camRot > limitCamRot Then
            camRot = limitCamRot
        ElseIf camRot < -limitCamRot Then
            camRot = -limitCamRot
        End If

        Dim deltaY As Single
        deltaY = (y - lasty) * 10
    End Sub

```



```

If bControl <> 0 Then
    UpdateCameraY deltaY
Else
    cameraDist = cameraDist - deltaY
    'Constrain "z-distance" to ensure that Visia is visible --
    "'z" here is as in graphics libraries, where +z is
    'by definition the direction the camera points in:
    If cameraDist < initZNear + fatnessAdj Then
        cameraDist = initZNear + fatnessAdj
    ElseIf cameraDist > initZFar Then
        cameraDist = initZFar
    End If
    SetCameraXZ centreX, centreZ, cameraDist, camRot
End If

lastx = x
lasty = y
End If
End Sub

Private Sub Avatar_OnInitialised()
    Debug.Print "Avatar_OnInitialised"
    bAvatarInitialised = True
    CheckReadyToStart
End Sub

Private Sub Avatar_OnDecreaseQueueLength(ByVal items As Long)

    Debug.Print "Avatar_OnDecreaseQueueLength " & items & _
        ", played: " & IPlayedCount

    If Not bAvatarInitialised Then
        Debug.Print "...but Avatar not initialised."
    End If

    If (awState = sPlaying) Or _
        (awState = sPlayingPaused) Or _
        (awState = sPlayingHalted) Then

        If awState = sPlaying Then
            IPlayedCount = IPlayedCount + 1
            UpdateSliderDisplay
        End If

        If IPlayedCount = IPlaylistSize Then
            ' ASSERT: awState = sPlaying
            SetAllPlayedThenDoReset
        ElseIf awState = sPlaying Then
            ' ASSERT: IPlayedCount <> IPlaylistSize
            FeedSignsToAvatar
        ElseIf items = 0 Then
            ' N.B. When a previous Avatar.StopAll is pending,
            ' i.e. when awState is sPlayingPaused or sPlayingHalted,
            ' we wait until the Avatar tells us it has completely
            ' flushed its internal queue:
            If awState = sPlayingPaused And IPlayedCount <> 0 Then
                SetStateReadyToResume
            Else
                SetStateReadyToStart
            End If
        End If
    End If
End Sub

Private Sub btnPlay_Click()
    Debug.Print "### Play Click."
    If IPlayedCount = IPlaylistSize Then
        ' ASSERT: awState = sReadyToResume
        SetAllPlayedThenDoReset
    End If
End Sub

```

```

Else
    ' ASSERT: awState in {sReadyToStart, sReadyToResume}
    StartPlaying
End If
End Sub

Private Sub btnPause_Click()
    Debug.Print "#### Pause Click."
    ' ASSERT: awState = sPlaying
    SetStatePlayingPaused
    Avatar.StopAll
End Sub

Private Sub btnStop_Click()
    Debug.Print "#### Stop Click."
    If awState = sPlaying Then
        SetStatePlayingHalted
        Avatar.StopAll
    ElseIf awState = sReadyToResume Then
        SetStateReadyToStart
    End If
End Sub

Private Sub Slider_Change()
    Dim INewPC As Long
    INewPC = Slider.Value
    If INewPC <> IPlayedCount Then
        Debug.Print "Slider change -- played (old): " & IPlayedCount & _
            ", played (new): " & INewPC
        IPlayedCount = INewPC
        If Slider.Value = 0 Then
            SetStateReadyToStart
        ElseIf awState = sReadyToStart Then
            SetStateReadyToResume
        End If
    End If
End Sub

Private Sub UserControl_DblClick()
    ' Safety net for use when the Visia Avatar Control lets us down
    ' -- treat this as a spoof OnDecreaseQueueLength():
    Debug.Print "SPOOF OnDecreaseQueueLength ..."
    Avatar_OnDecreaseQueueLength (0)
End Sub

Private Sub UserControl_Initialize()
    Debug.Print ""
    Debug.Print "UEAVisiaDummyWrapper - UserControl_Initialize #####"
    SetStateSettingUp
    StartVisiaAvatarControl
End Sub

Private Sub UserControl_Terminate()
    Debug.Print "UEAVisiaDummyWrapper - UserControl_Terminate"
    If awState = sPlaying Then
        SetStatePlayingHalted
        Avatar.StopAll
        Debug.Print "... Stop call made to Visia avatar"
    End If
End Sub

Public Sub AddToPlayList(sMotionFile As String)
    Attribute AddToPlayList.VB_Description = "Pass in a single motion file"
    ' Resize array to add new item
    Debug.Print "PL Size: " & IPlaylistSize & _
        ", now adding mocap: " & sMotionFile
    ReDim Preserve arsPlaylist(0 To IPlaylistSize)
    arsPlaylist(IPlaylistSize) = sMotionFile
    IPlaylistSize = IPlaylistSize + 1
    UpdateSliderLimits

```

```
    CheckReadyToStart
End Sub
```

```
Public Sub ReadSiGML(xmlDoc As DOMDocument30)
Attribute ReadSiGML.VB_Description = "Pass in a motion file play-list as an XML document"
    MsgBox xmlDoc.xml
```

```
On Error GoTo ReadSiGML_Error
    Dim nodeList As IXMLDOMNodeList
    Set nodeList = xmlDoc.selectNodes("gml/gmlsign/lexsign")
    Dim l As Long
    Dim nodeLexSign As IXMLDOMNode
    Dim nodeMoCapData As IXMLDOMNode
    Dim attFileName As IXMLDOMAttribute
    For l = 0 To nodeList.length - 1
        Set nodeLexSign = nodeList.Item(l)
        Set nodeMoCapData = nodeLexSign.selectSingleNode("mocapdata")
        Set attFileName = nodeMoCapData.Attributes.getNamedItem("filename")
        Debug.Print attFileName.Text
        ReDim Preserve arsPlaylist(0 To IPlaylistSize)
        arsPlaylist(IPlaylistSize) = attFileName.Text & ".all"
        IPlaylistSize = IPlaylistSize + 1
    Next l
    Set nodeList = Nothing
    UpdateSliderLimits
    CheckReadyToStart
Exit Sub
```

```
ReadSiGML_Error:
    MsgBox "ReadSiGML Error: " & Err.Description
End Sub
```

(A) Specification webpage

Preface

This is the specification for a webpage containing a combination of information in text and information signed by an avatar. The specification has been compiled by the three organizations participating in ViSiCAST that are most involved with deaf people: IvD, RNID and UH.

User's starting-point

Internet-explorer has been started up. The user types in the address of the ViSiCAST weather forecast, or uses a bookmark, or follows a link (from pages known to deaf people, or from the ViSiCAST-page).

The pages

Now three versions are possible:

1. simultaneous presentation of text and avatar (inside the browser)
2. separate presentation of the text and avatar (avatar still in the browser)
3. avatar in separate window.

The first and the second options are rather equal. At RNID and at IvD we prefer option 1. Thomas from UH did not express a preference between 1 and 2. However, he stressed that one very important difference between the Vcom3D software (<http://www.signingavatar.com/>) and the Televirtual software was up to now the possibilities to have a full-screen avatar and to resize the avatar. So, an important question is: *can the avatar be resized when it sits inside the webpage, or can the size of the avatar be set relative to the window of the browser?*

General remarks about presentation of the avatar and the text are given first. Then sketches of the pages in the three options are given.




General design issues

- quiet lay-out
- not many colours, preferably no more than three colours
- no moving objects other than where the focus of attention should be
- clear contrast in text and icons
- use only icons leave absolutely no doubt about the function of the button

“Avatar player”

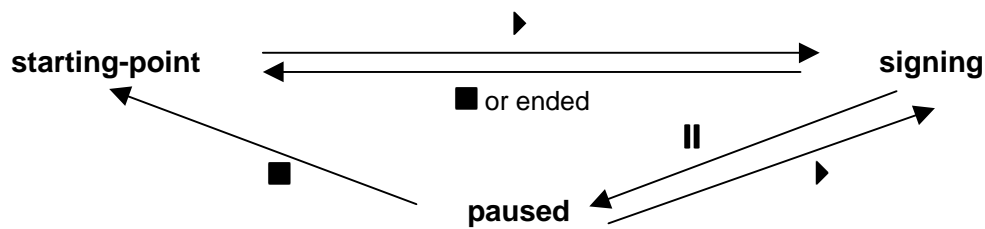
The user should be able to control the avatar with three buttons: Play-Pause-Stop. In general, the ‘look and feel’ of the avatar player should be like the well-known media players (QuickTime, Windows Media Player). Below, I described the look-and-feel that we have in mind.

The states that the player can be in are: starting-point, signing, and paused. In these states, the buttons should be presented plain, flickering, grey etc. according to the table below.

	 PLAY	 PAUSE	 STOP	avatar
starting-point	flickering	disabled	disabled or pressed	ready to start signing from the beginning
signing	pressed	enabled	enabled	moving

paused	enabled	pressed	enabled	ready to continue signing where it has been paused
---------------	---------	---------	---------	--

The possible transitions between the states are:



If there is a problem in realizing this functionality and appearance at the same time, just let me know and we can discuss which wishes can be dropped and which wishes can be realized.

Question: could it also be made possible to click on the avatar itself to start it (besides the Play-button)? And stopping or pausing?

For the size of the avatar, we prefer 15cm x 15cm. For the buttons, we prefer 2cm x 2cm. Do these sizes vary when the resolution varies across users? In that case care has to be taken that most users see the mentioned sized, and not only the users with the most advanced technology.

Text frame


A sans-serif font, not too small, is preferred. For example 12 point Arial/Helvetica. In case the fontsize can only be specified relative to what the user chose for himself: 1 point extra. Furthermore, the length of the lines should be around 40 to 60 characters. Preferably no scrollbar. An example weather forecast is below.

Weerbericht van donderdag 17 augustus 2000, 17:34 uur

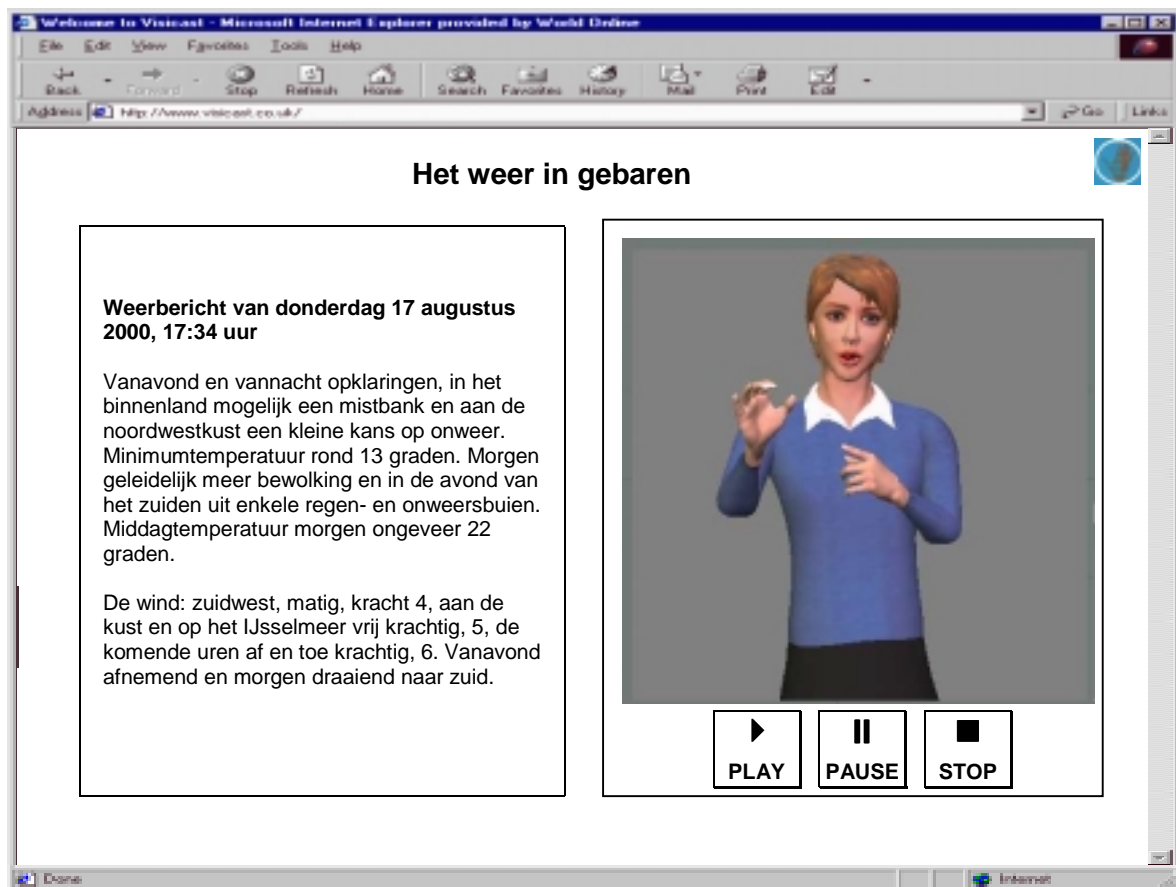
Vanavond en vannacht opklaringen, in het binnenland mogelijk een mistbank en aan de noordwestkust een kleine kans op onweer. Minimumtemperatuur rond 13 graden. Morgen geleidelijk meer bewolking en in de avond van het zuiden uit enkele regen- en onweersbuien. Middagtemperatuur morgen ongeveer 22 graden.

De wind: zuidwest, matig, kracht 4, aan de kust en op het IJsselmeer vrij krachtig, 5, de komende uren af en toe krachtig, 6. Vanavond afnemend en morgen draaiend naar zuid.

Page sketches

- In options 1 and 2 it should be guaranteed for all users (different screen-sizes, different resolutions) that the size of the avatar is minimally 11x11cm and that the window fits on the screen.
- The -logo is a link to the ViSiCAST webpage, but the link does not appear as a button (in order to avoid confusion with any control buttons).
- Below I did not yet add a sidebar for the movie, but after talking to Thomas from UH, I am convinced it should be there after all, if possible. Both for navigation and for orientation in the “movie”.

1. Simultaneous presentation in one page



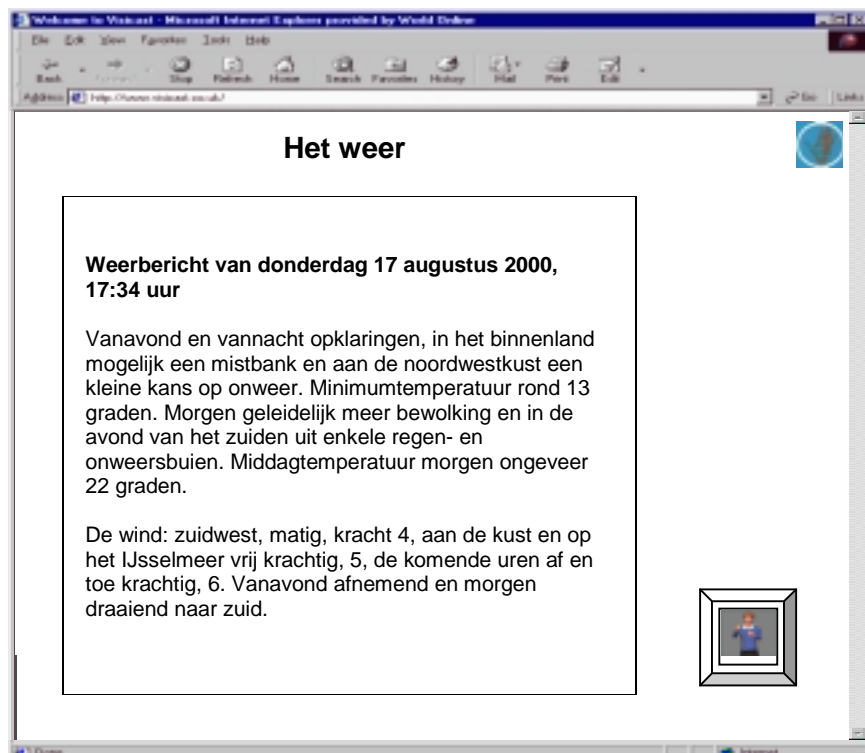
This window should fit on the screen, with a reasonably sized avatar. If this means squeezing the text box, this is ok as long as the lines contain minimally about 35 characters, and minimally two third of the text is visible before scrolling (vertically).

2. Separate presentation in two pages

The avatar page appears first:



With a button in the lower right corner you can go to the text version of the forecast:



And from the text version you can switch back to the avatar with a very similar button. Also in general, the layout of the two pages is very similar: similar sizes of the avatar and the text box, and same colours and same border.

The pictograms on the buttons must be very clear. For the pictogram meaning “text”, it is best to use something that is used in other software as well: lines, lines of different lengths, or lines + one big letter like the icon for “textbox” in MS-Word. As pictogram meaning “avatar” or “signs” it is tempting to use a pictogram that usually means movie or player, but the semantics would not be really correct. Sometimes a pictogram of one or two hands is suggested, but a common argument against this solution is that mimics, mouthing, and body movements are essential and substantial parts of the language. Therefore, something that looks like the avatar would be best. It could be simplified.

3. *Separate presentation in one page and one avatar-window*

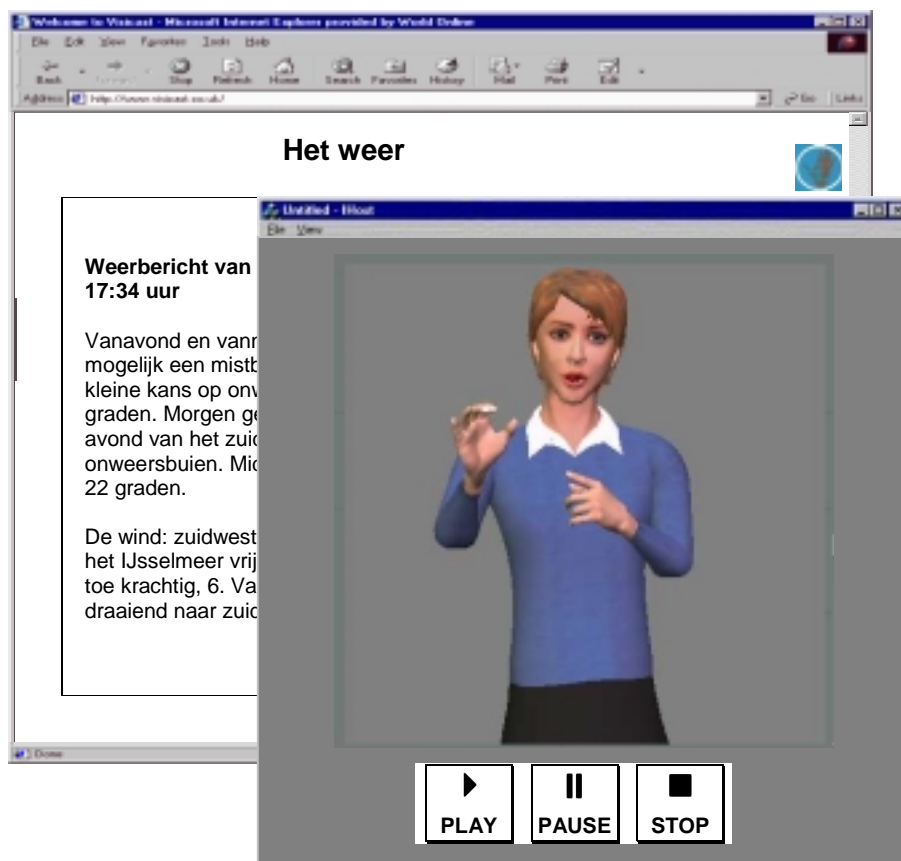
The text page (like in option 2) appears first.

The separate avatar-window pops up automatically, over the browser window.

The user can minimize or close the avatar with the usual Windows-controls.

Clicking the avatar button makes the avatar window re-appear (either restore the window from the taskbar or pop up again).

The user can move and resize the avatar window like any window. Resizing the window also resizes the avatar.



We ask this, because in the Dan-version that Sanja gave us in March, the avatar could be resized. Is this still possible with controls in the avatar window? Otherwise these player controls have to go on the webpage. Then, the positioning of the avatar window becomes very important. Can the positioning of new windows be controlled from the webpage? A separate “mini-window” with only the player controls is another solution.

(B) Model for weather forecasts with mapping

(without list of words)

Preface

This is the description of the language model used to implement the Creator software for weather forecasts in Sign Language of the Netherlands. Originally, this description was in Dutch, but in order to be informative to a more readers the English translation of it is given here rather than the Dutch version. Apart from that, this is (together with the list of words) exactly the document used for communication between the partners compiling the language model and the partners implementing it.

The sentences in black are the sentence forms that can be used in the weather forecast. These are not full sentences, but rather fill-in sentences, so-called 'patterns'. The words in the patterns that are between square brackets refer to a category of words that can be filled in at this position. Those categories can be found in the separate appendix "List of words".

In blue, it is indicated how the corresponding sentence is converted to signs. Exceptions to certain rules for the conversion are given in red. Where there is no systematic variation in the patterns, the blue instructions point to the list of words, which is in fact a straight forward "look-up table".

introduction sentence

see list of words, section "FIXED PHRASES OF PATTERNS"

1. *The weather forecast of the KNMI, drawn up on [WEEKDAY] [NUMBER][MONTH], and valid till midnight.*
2. *The weather forecast of the KNMI, drawn up on [WEEKDAY] [NUMBER][MONTH], and valid till midnight tomorrow.*
3. *Summary forecasts [WEEKDAY] [NUMBER][MONTH].*

sentence in between

see list of words, section "FIXED PHRASES OF PATTERNS"

The further prospects of [WEEKDAY] up to and including [WEEKDAY].

outlook for the weather in general

1. [WEATHER CONDITION]
= sign[WEATHER CONDITION]
2. [PLACE] [WEATHER CONDITION]
= sign[PLACE] sign[WEATHER CONDITION]
if [PLACE]= elsewhere / here and there / at many places / locally
then sign[WEATHER CONDITION] sign[PLACE]
3. [TIME] [WEATHER CONDITION]
= sign[TIME] sign[WEATHER CONDITION]
if [TIME]= now and then
then sign[WEATHER CONDITION] sign[TIME]
4. [TIME] [PLACE] [WEATHER CONDITION]
= sign[TIME] sign[PLACE] sign[WEATHER CONDITION]
if [PLACE]= elders / hier en daar / op veel plaatsen / placeelijk
then sign[TIME] sign[WEATHER CONDITION] sign[PLACE]
if [TIME]= af en toe
then sign[PLACE] sign[WEATHER CONDITION] sign[TIME]

5. [TEMPERATURE]
separate temperature-expressions: see further down
6. [PLACE] [TEMPERATURE]
= sign[PLACE] sign[TEMPERATURE]
7. [TIME] [TEMPERATURE]
= sign[TIME] sign[TEMPERATURE]
8. [TIME] [PLACE] [TEMPERATURE]
= sign[TIME] sign[PLACE] sign[TEMPERATURE]

outlook for the wind

1. [WIND-FORCE]
= sign[WIND-FORCE]
2. [WIND-FORCE] [WIND-DIRECTION]
= sign[WIND-FORCE] sign[WIND-DIRECTION]
3. [PLACE] [WIND-FORCE]
= sign[PLACE] sign[WIND-FORCE]
4. [TIME] [WIND-FORCE]
= sign[TIME] sign[WIND-FORCE]
5. [PLACE] [WIND-FORCE] [WIND-DIRECTION]
= sign[PLACE] sign[WIND-FORCE] sign[WIND-DIRECTION]
if [WIND-DIRECTION]= *seawind*
then sign[PLACE] sign[WIND-DIRECTION] sign[WIND-FORCE]
6. [TIME] [WIND-FORCE] [WIND-DIRECTION]
= sign[TIME] sign[WIND-FORCE] sign[WIND-DIRECTION]
if [WIND-DIRECTION]= *seawind*
then sign[TIME] sign[WIND-DIRECTION] sign[WIND-FORCE]
7. [TIME] [PLACE] [WIND-FORCE]
= sign[TIME] sign[PLACE] sign[WIND-FORCE]
8. [TIME] [PLACE] [WIND-FORCE] [WIND-DIRECTION]
= sign[TIME] sign[PLACE] sign[WIND-FORCE] sign[WIND-DIRECTION]

composed weather conditions

see list of words, section "WEATHER CONDITIONS"

1. [WEATHER CONDITION] *and* [WEATHER CONDITION]
2. [WEATHER CONDITION] *followed by* [WEATHER CONDITION]
3. [WEATHER CONDITION] *interspersed by* [WEATHER CONDITION]

composed places

1. [PLACE] *and* [PLACE]
= sign[PLACE] sign[PLACE]
2. ~~*from* [PLACE] *to* [PLACE]~~

composed times

see list of words, section "TIME"

1. *from* [TIME] *to* [TIME]
2. [TIME] *of* [TIME]
3. [TIME] *and* [TIME]

composed wind directions

[WIND DIRECTION] *turning to* [WIND DIRECTION]
combinations enumerated in list of words

expressions of wind force

see list of words, section “WIND-FORCE”

4. [WIND-FORCE]
5. [WIND-FORCE], *wind-force* [NUMBER]
6. *wind-force* [NUMBER]
7. *wind-force* [NUMBER] *of less*
8. *wind-force* [NUMBER] *or more*
9. *wind-force* [NUMBER] *to* [NUMBER]

expressions of temperature

see list of words, section “TEMPERATURE”

1. temperature about [NUMBER] degrees
2. ‘highest temperature in the afternoon’ about [NUMBER] degrees
3. minimum temperature about [NUMBER] degrees
4. maximum temperature about [NUMBER] degrees
- ~~5. temperature around freezing point~~
- ~~6. ‘highest temperature in the afternoon’ around freezing point~~
- ~~7. minimum temperature around freezing point~~
- ~~8. maximum temperature around freezing point~~
9. higher temperature
10. lower temperature
11. it gets less cold/milder
12. it gets less warm/cooler

(C) List of words

(belonging to the model for weather forecasts)

Preface

This is the list needed to convert single Dutch words (or short phrases) to SLN-signs. Originally, this is a Dutch document. In order to be accessible for more readers, the English translation is provided rather than the Dutch version. Apart from that, this is (together with the model-mapping) exactly the document used for communication between the partner compiling the language model and the partner implementing it.

The signs for the words/phrases are in files with identical names, unless stated otherwise (extensions .all).

Some phrases are composed from several SLN-files. In column on the right, you see the names of the files, separated by '+'. (Spaces are no separations; spaces occur in the file names!)

Words striked-through, were not correctly recorded. These Dutch words must be dropped from the list. Underlined words or file names have been changed since last time.

Dutch phrase:

names SLN-files, *.all (if not mentioned:
identical to the Dutch phrase on the left):

---- [weekday]: ----

1. *monday*
2. *tuesday*
3. *wednesday*
4. *thursday*
5. *friday*
6. *saterdag*
7. *sunday*

---- [month]: ----

8. *January*
9. *February*
10. *March*
11. *April*
12. *May*
13. *June*
14. *July*
15. *August*
16. *September*
17. *October*
18. *November*
19. *December*

---- [time]: ----

20. *now and then*afwisselend21. *during the day*22. ~~*geruime tijd*~~23. *in the evening*24. *in the afternoon*25. *during the night*26. *latter part of the night*27. *in the morning*28. *in the early morning*29. *in het weekend*weekend30. *considerable time*31. *tomorrow*32. *tomorrow by day*33. *tomorrow evening*34. *tomorrow afternoon*35. *tomorrow midnight*36. *tomorrow morning*37. *after the night*38. *after the weekend*39. ~~*after midnight*~~

?

40. *by day*41. *around midnight*42. *at the end of the evening*43. ~~*time to time*~~44. *tonight*45. *today*46. *this afternoon*47. *this night*48. *(from) ... to ...*... + van [tijd] tot [tijd]-A + ... +
van [tijd] tot [tijd]-B49. *... and ...*... + ... + [tijd] en [tijd]-def50. *... or ...*... + [tijd] of [tijd]-def + ...

---- [place]: ----

51. *at the coast*52. *at sea*53. *above land*54. *on sea*55. *Drente*56. *elsewhere*57. ~~*Flevoland*~~

?

58. *Friesland*59. *Gelderland*60. *Groningen*61. *here and there*62. *in the littorals*63. *inland*64. *in the whole country*65. *in the littoral*66. *in the centre of the country*67. *in the north*

68. *in the north-east*
69. *in the north-west*
70. *in the east*
71. *in the west*
72. *in the south*
73. *in the south-east*
74. *in the south-west*
75. *off shore*
76. *Limburg*
77. *Noord-Brabant*
78. *Noord-Holland*
79. *at many places*
80. *everywhere*
81. *Overijssel*
82. *locally*
83. *at the north of the big rivers*
84. *at the south of the big rivers*
85. *Utrecht*
86. *from the coast*
87. *from the sea*
88. *from the north*
89. *from the north-east*
90. *from the north-west*
91. *from the east*
92. *from the west*
93. *from the south*
94. *from the south-east*
95. *from the south-west*
96. *Zeeland*
97. *Zuid-Holland*

zuidwest

---- [temperature]: ----

(numbers further down the list)

98. *higher temperature*
99. *lower temperature*
100. *it's getting warmer*
101. *it's getting colder*
102. ~~*temperature rond het vriespunt*~~
103. ~~*middagtemperature rond het vriespunt*~~
104. ~~*minimumtemperature rond het vriespunt*~~
105. ~~*maximumtemperature rond het vriespunt*~~
106. *temperature about ... degrees*
107. *middagtemperature about ... degrees*
108. *minimumtemperature about ... degrees*
109. *maximumtemperature about ... degrees*

temperatuur + rond het vriespunt
 middagtemperatuur + rond het vriespunt
 minimumtemperatuur + rond het vriespunt
 maximumtemperatuur + rond het vriespunt
temperatuur ongeveer getal graden-A + ... +
temperatuur ongeveer getal graden-B
middagtemperatuur ongeveer getal graden-A + ...
+ middagtemperatuur ongeveer getal graden-B
minimumtemperatuur ongeveer getal graden-A +
... + minimumtemperatuur ongeveer getal graden-B
maximumtemperatuur ongeveer getal graden-A +
... + maximumtemperatuur ongeveer getal graden-B

---- [wind-force]: ----

110. *subsiding wind*
111. *hard wind*
112. *high wind*

veel wind
veel wind

113. *moderate wind*

~~114. *storm*~~

~~storm~~

115. *stormy*

storm

116. *freshening wind*

117. *much wind*

118. *pretty high wind*

matige wind

119. *few wind*

120. *light wind*

121. *heavy gusts of wind*

windstoten

122. *..., wind-force ...*

... + windkracht + ...

123. *wind-force ...*

windkracht + ...

124. *wind-force ... or more*

windkracht of meer-A + ... + windkracht of meer-B

125. *wind-force ... or less*

windkracht of minder-A + ... + windkracht of minder-B

126. *wind-force ... to ...*

windkracht tot-A + ... + windkracht tot-B + ...

---- [wind-direction]: ----

127. *N*

128. *NE*

129. *NW*

130. *E*

131. *from various directions*

132. *changing*

133. *W*

134. *seawind*

135. *S*

136. *SE*

137. *SW*

composed wind-directions (... *till* ..., only 45 degrees one 1 order, order doesn't matter):

138. *N till NE*

139. *N till NW*

140. *E till NE*

141. *E till SE*

142. *W till NW*

143. *W till SW*

144. *S till SE*

145. *S till SW*

composed wind-directions (... *turning to* ..., 45 degrees, clockwise):

146. *N turning to NE*

noord naar noordoost

147. *NE turning to E*

noordoost naar oost

148. *NW turning to N*

noordwest naar noord

149. *E turning to SE*

oost naar zuidoost

150. *W turning to NW*

west naar noordwest

151. *S turning to SW*

zuid naar zuidwest

152. *SE turning to S*

zuidoost naar zuid

153. *SW turning to W*

zuidwest naar west

composed wind-directions (... *turning to* ..., 45 degrees, counter-clockwise):

154. *N turning to NW*

noord naar noordwest

155. *NE turning to N*

noordoost naar noord

156. NW turning to W	<u>noordwest naar west</u>
157. E turning to NE	<u>oost naar noordoost</u>
158. W turning to SW	<u>west naar zuidwest</u>
159. S turning to SE	<u>zuid naar zuidoost</u>
160. SE turning to E	<u>zuidoost naar oost</u>
161. SW turning to S	<u>zuidwest naar zuid</u>

composed wind-directions (... turning to ..., 90 degrees, clockwise):

162. N turning to E	<u>noord naar oost</u>
163. NE turning to SE	
164. NW turning to NE	
165. E turning to S	<u>oost naar zuid</u>
166. W turning to N	<u>west naar noord</u>
167. S turning to W	<u>zuid naar west</u>
168. SE turning to SW	
169. SW turning to NW	

composed wind-directions (... turning to ..., 90 degrees, counter-clockwise):

170. N turning to W	<u>noord naar west</u>
171. NE turning to NW	
172. NW turning to SW	
173. E turning to N	<u>oost naar noord</u>
174. W turning to S	<u>west naar zuid</u>
175. S turning to E	<u>zuid naar oost</u>
176. SE turning to NE	
177. SW turning to SE	

---- [weather condition]: ----

178. freezing	
179. freezing of wet patches on the road	
180. cloudy	
181. the sun is shining	
182. thick to dense fog-banks	zeer dichte misbanken
183. thick fog	
184. dry	
185. calm wintry weather	
186. raw	
187. hail	
188. clear sky	
189. slippery	
190. autumn weather	herfst + weer
191. glazed frost	
192. cold	
193. low clouds	
194. spring weather	lente + weer
195. moderate to severe frost	
196. moderate frost	
197. less steady	
198. fog	
199. fog-banks	
200. drizzling rain	
201. sleet	

- 202. *thunder*
- 203. *thunder-shower*
- 204. *bright periods*
- 205. *rain showers*
- 206. *mild weather*
- 207. *veil clouds*
- 208. *snow*
- 209. *storm*
- 210. *severe frost*
- 211. *gathering clouds*
- 212. *frost*
- 213. *pretty sunny*
- 214. *few clouds*
- 215. *winter's showers*
- 216. *wintery weather*
- 217. *varying clouded*
- 218. *changeable*
- 219. *cloud-layers*
- 220. *minima*
- 221. *dense fog-banks*
- 222. *summer weather*
- 223. *sunny*
- 224. *heavy clouded*
- 225. *... and ...*
- 226. *... followed by ...*
- 227. *... interspersed with ...*

... ook ...
... dan-gevolgd door ...
... .. afwisselend

---- [number]: ----

- | | | |
|------|------|--------------|
| 228. | 0 | nr0 |
| 229. | 1 | nr1 |
| 230. | 2 | nr2 |
| 231. | 3 | nr3 |
| 232. | 4 | nr4 |
| 233. | 5 | nr5 |
| 234. | 6 | nr6 |
| 235. | 7 | nr7 |
| 236. | 8 | nr8 |
| 237. | 9 | nr9 |
| 238. | 10 | nr10 |
| 239. | 11 | nr11 |
| 240. | 12 | nr12 |
| 241. | 13 | nr13 |
| 242. | 14 | nr14 |
| 243. | 15 | nr15 |
| 244. | 16 | nr16 |
| 245. | 17 | nr17 |
| 246. | 18 | nr18 |
| 247. | 19 | nr19 |
| 248. | 20 | nr20 |
| 249. | 21 | nr1 + nr20 |
| 250. | 22 | nr22 |
| 251. | 23 | nr3 + nr20 |
| 252. | 24 | nr4 + nr20 |
| 253. | 2... | nr... + nr20 |

254.	30	nr30
255.	33	nr33
256.	40	nr40
257.	44	nr44
258.	50	nr50
259.	55	nr55
260.	60	nr60
261.	66	nr66
262.	70	nr70
263.	77	nr77
264.	80	nr80
265.	88	nr88
266.	90	nr90
267.	99	nr99
268.	100	nr100
269.	200	nr2 + nr100
270.	1000	nr1000
271.	2000	nr2000
272.	other numbers < 100	[singles] + [tens]
273.	other numbers > 100 , < 1000	[hundreds] + nr100 + [singles] + [tens]
274.	other numbers > 2000 , < 2099 (years)	nr2000 + [singles] + [tens]
275.	numbers < 0	min + ...
---- fixed phrases of patterns: ----		
276.	<i>The weather forecast of the KNMI, drawn up on ... and valid till midnight.</i>	<u>KNMI + ... + tot-boog + middernacht</u>
277.	<i>The weather forecast of the KNMI, drawn up on ... and valid till midnight tomorrow.</i>	<u>KNMI + ... + tot-boog + morgenmiddernacht</u>
278.	<i>Summary forecasts ...</i>	<u>... + overzicht verwachtingen</u>
279.	<i>The further prospects of ... up to and including ...</i>	<u>... + tot-verticaal + ... + weer + verwachtingen</u>

(D) ViSiCAST Weather Forecast Creator User Manual

1 Introduction

Deliverable D2-1 of ViSiCAST Work Package 2, is a prototype of an Internet Browser Plug-in, which provides signing of a weather forecast by an Avatar that is part of a web page. This document describes a software tool that was written to allow the dynamic creation of a ViSiCAST Weather Forecast web page.

2 Weather Phrases

The identification of all the various phrases that constitute a weather forecast was compiled by Margriet Verlinden of IvD, [1]. The structure (grammar), of a weather forecast, was also compiled by Margriet Verlinden, [2].

3 Overview

A utility has been developed that allowed a user to create a Weather Forecast using pre-defined natural language standard phrases and parameters, [1], [2]. A user is able to select phrases and appropriate parameters to create the sentences required. Each sentence is then parsed and each phrase mapped onto one or more motion files that are added to a play-list that constitutes part of the Weather Report.

The utility allows the creation of new sentences to add to the play-list, the editing of existing sentences in the play-list, the deletion of a sentence from the play-list and the re-ordering of sentences in the play-list.

The utility creates an output HTML file that allows the Weather Forecast to be displayed as text and played, and therefore, signed, by an Avatar on a Web Page, [3].

The utility is built with multi-language capability, as it supports, English, Dutch and German.

4 Design

4.1 DTD

A DTD was designed that describes the structure of a Weather Forecast. It includes a unique identifier for each Weather Forecast phrase, [4], [5].

4.2 Language DLLs

A Language DLL for each supported language (currently English, Dutch and German) has been built that contain resource strings and dialogs, for the User Interface.

4.3 XML Data File

An XML data file was created that has an ID for each weather phrase, its natural language description and a list of the motion files required to sign that phrase. An example of a phrase follows:

```
<IntroductionSummary lang="Dutch">
  <Phrase>
    <ID>summary_forecasts</ID>
    <Text>Overzicht verwachtingen</Text>
    <MotionFiles>
      <FileName>SLN_overzicht_verwachtingen.all</FileName>
    </MotionFiles>
  </Phrase>
</IntroductionSummary>
```

Note that the example phrase is enclosed in a tag that defines what category of phrase it is, in this case an 'IntroductionSummary' phrase; other examples of types of phrases are 'Month', 'Time' and 'Place'. Each category of phrase has an attribute named 'lang' which denotes the language of all the phrases contained within it. The ID of the phrase

corresponds to a value defined in the DTD. The 'Text' of the phrase will be displayed to the user in the appropriate combo-box of the Weather Forecast Creator. Finally, a list of one or more motion files required to sign the phrase in the appropriate sign language, which in the above example is SLN as the language of the phrase is Dutch.

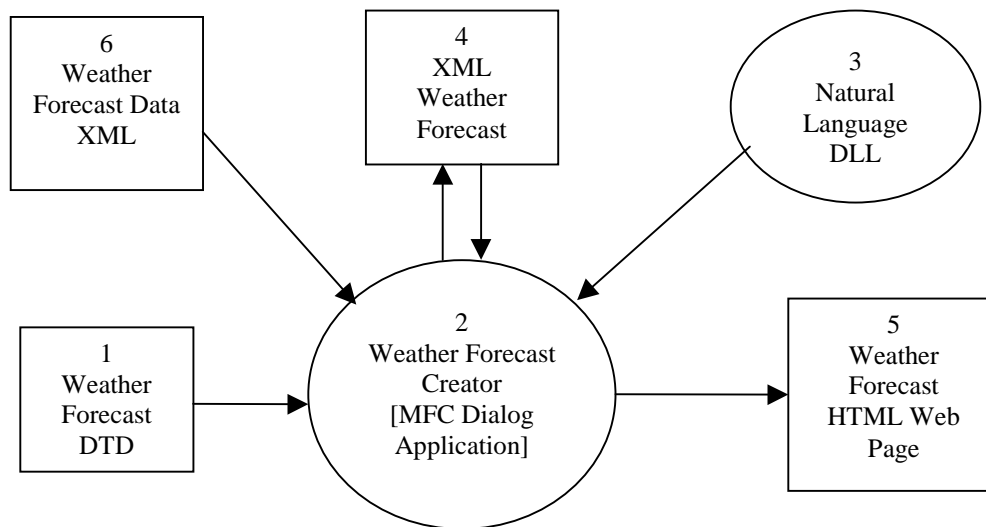
In this example the category of phrase has only one phrase, whereas most categories have several phrases, for example the 'Time' phrase category.

Note also that there is also a phrase category called 'IntroductionSummary' with its 'lang' attribute equal to 'English'. This design allows for different phrases for each language to be used for each category.

4.4 Output XML

The data for a Weather Forecast is stored in a temporary XML file whose structure is defined by the Weather Forecast DTD. The output XML file contains an ID for each weather phrase chosen. This ID is used firstly, to look up the corresponding natural language phrase and secondly, to look up the motion file or files required to be played for that phrase.

4.5 Architecture



The Weather Forecast Creator program uses the Weather Forecast Data XML to fill in choices in the various combo-boxes. It then uses the DTD to validate the choices made by the user and creates an XML file. This XML file is then processed into HTML, using the Weather Forecast Data XML to select the appropriate text, and the names of the required motion file or files, and then saved with the name and location specified by the user.

4.6 User Interface

The Weather Forecast dialog has a tab for each category of sentence that can exist in the Weather Forecast, e.g. General and Temperature. A list of the sentences that make up the Weather Report (in natural language) is always displayed.

There are various buttons that are described below:

- an 'Add' button, to add the details in the currently active tab to the list
- an 'Edit' button to allow the user to change an existing sentence in the list
- an 'Update' button, that updates a sentence that has been edited
- a 'Delete' button, to remove the currently selected list sentence
- 'Move Up' and 'Move Down' buttons to alter the ordering of sentences
- a 'Create Web Page' button which will produce an HTML file with the name and location specified by the user.

5 Implementation

5.1 Implementation Tools

The utility was built as a Visual C++ MFC dialog application using Visual C++ v6 (SP4). Access to XML functionality was by use of the '#import' of the 'MsXml3.Dll' file which created wrapper classes of the interfaces exposed by the dll.

The name of the executable is 'Weather Forecast Creator MkII.exe'.

5.2 XML Data File

The name of the XML data file is 'WeatherForecastData.xml' and this should always reside in the same folder as 'Weather Forecast Creator MkII.exe'.

5.3 DTD

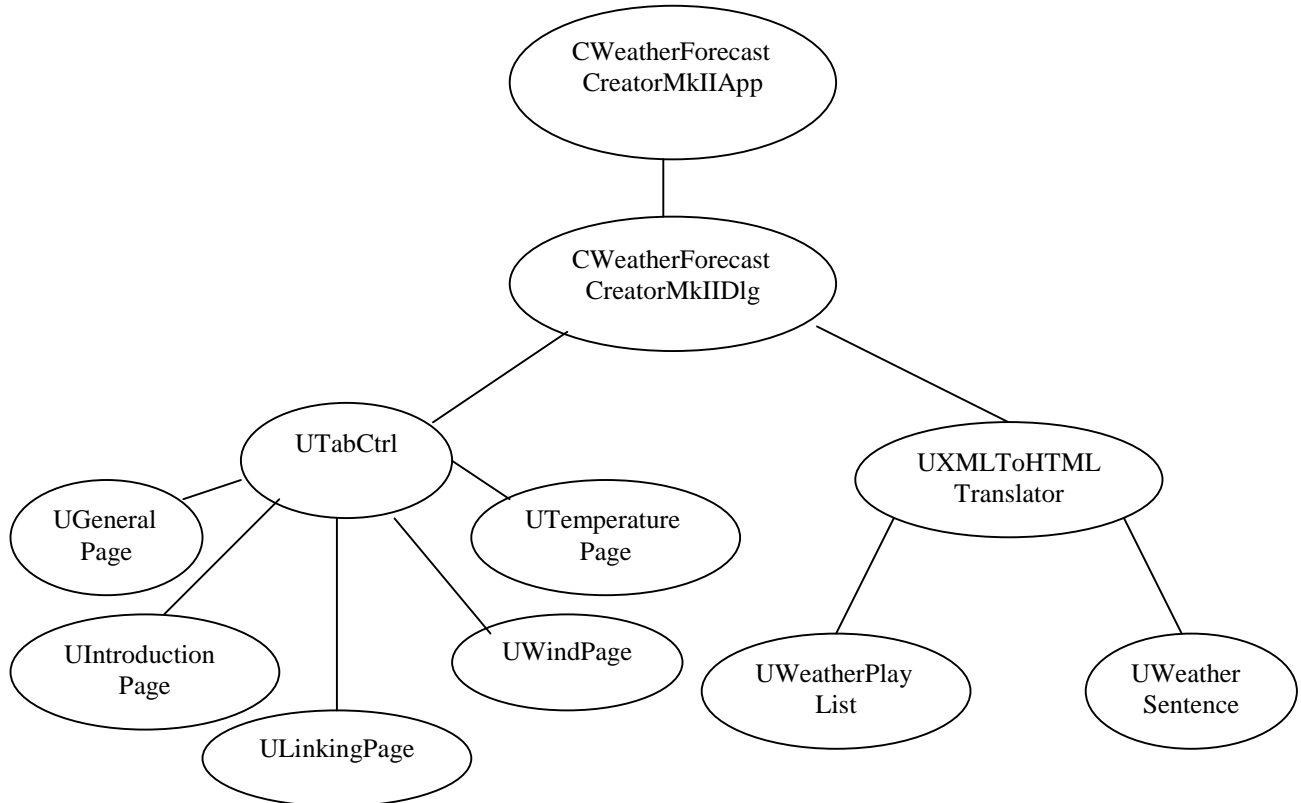
The name of the DTD is 'Weather Forecast.dtd' and this should always reside in the same folder as 'Weather Forecast Creator MkII.exe'.

5.4 Language DLLs

There are three natural language dlls, one each for English, Dutch and German. The dll file of the language being used must always reside in the same folder as 'Weather Forecast Creator MkII.exe'.

5.5 Class Diagram

The diagram below depicts the classes implemented and the classes used by a particular class. Each class is discussed below.



5.5.1 Application Class

This class, CWeatherForecastCreatorMkIIApp, is defined in 'Weather Forecast Creator MkII.h' and implemented in file 'Weather Forecast Creator MkII.cpp'.

This is a standard AppWizard generated dialog application. Code added to that produced by the wizard, mainly to the 'InitInstance' function, was:

- the call to the CoInitialize function to initialise the COM libraries
- the call to the SetRegistryKey function
- code to handle the loading of the appropriate language resource DLL

The call to 'CoUninitialize' function was added to the 'ExitInstance' function.

5.5.2 Weather Creator Dialog Class

This class, CWeatherForecastCreatorMkIIDlg, is defined in 'Weather Forecast Creator MkIIDlg.h' and implemented in the files 'Weather Forecast Creator MkIIDlg.cpp', 'Weather Forecast Creator MkIIDlg_General.cpp', 'Weather Forecast Creator MkIIDlg_Introduction.cpp', 'Weather Forecast Creator MkIIDlg_Linking.cpp', 'Weather Forecast Creator MkIIDlg_Temperature.cpp', and 'Weather Forecast Creator MkIIDlg_Wind.cpp'.

This class is derived from the MFC CDialog class. Important points to note regarding this class are:

- It uses the UTabCtrl class which is discussed below
- It uses two IXMLDOMDocument2Ptr class member variables, one for the output XML document and one for the XML data document. This class and all other XML classes are defined in the file 'msxml3.tlh' which, along with the file 'msxml3.tli', is created by the '#import <msxml3.dll>' directive in the 'stdafx.h' file.
- It provides the function 'GetOutputText' which when given the ID of a phrase queries the XML data document for the output text for the phrase with that ID (i.e. the value of the <Text> tag); this function is used by the Weather Sentence class described below
- It provides the function 'GetMotionFiles' which when given the ID of a phrase queries the XML data document for all the motion files required by that phrase with that ID (i.e. the value of all the <FileName> tags in the < MotionFiles> tag); this function is used by the Weather Playlist class described below
- It creates an XML file called 'Weather.xml', which is always in the same directory as 'Weather Forecast Creator MkII.exe', which is over-written each time the user clicks on the 'Create HTML File' button. This file is then passed to the XML to HTML translator class described below.

5.5.3 Tab Control

This class, UTabCtrl, is defined in TabCtrl.h and implemented in 'TabCtrl.cpp'.

This class is a slightly amended version of an example of a class derived from the MFC class CTabCtrl downloaded from the internet. It creates and contains the five dialogs described below in separate tabs and implements the switching from one tab to another.

5.5.4 General Page

This class, UGeneralPage, is defined in 'GeneralPage.h' and implemented in 'GeneralPage.cpp'.

This class is derived from the MFC class CDialog. It implements the behaviour of the user interface of the General tab.

5.5.5 Introduction Page

This class, UIntroductionPage, is defined in 'IntroductionPage.h' and implemented in 'IntroductionPage.cpp'.

This class is derived from the MFC class CDialog. It implements the behaviour of the user interface of the Introduction tab.

5.5.6 Linking Page

This class, ULinkingPage, is defined in 'LinkingPage.h' and implemented in 'LinkingPage.cpp'.

This class is derived from the MFC class CDialog. It implements the behaviour of the user interface of the Linking tab.

5.5.7 Wind Page

This class, UWindPage, is defined in 'WindPage.h' and implemented in 'WindPage.cpp'.

This class is derived from the MFC class CDialog. It implements the behaviour of the user interface of the Wind tab.

5.5.8 Temperature Page

This class, UTemperaturePage, is defined in 'TemperaturePage.h' and implemented in 'TemperaturePage.cpp'.

This class is derived from the MFC class CDialog. It implements the behaviour of the user interface of the Temperature tab.

5.5.9 XML To HTML Translator

This class, UXMLToHTMLTranslator, is defined in 'XMLToHTMLTranslator.h' and implemented in 'XMLToHTMLTranslator.cpp'.

This class has a single public member function, 'DoTranslation', which is called by the Weather Creator dialog class. It takes the XML file, whose filename is passed in as a parameter to the constructor, and creates an HTML file, whose name is also passed in as a parameter to the constructor.

It uses the class 'UWeatherSentence' to help in the creation of the textual representation of the weather sentences. See the member function 'ExtractIntroduction' for an example of this.

It uses the class 'UWeatherPlayList' to help in the creation of the Javascript for adding motion files to a play-list. See the member function 'GetIntroductionFiles' for an example of this.

5.5.10 Weather Sentence

This class, UWeatherSentence, is defined in 'WeatherSentence.h' and implemented in 'WeatherSentence.cpp'.

It has a single public member function, 'GetWeatherSentence' that is called by the Weather Creator dialog class and the XML to HTML Translator class. It uses the function 'GetOutputText' implemented in the 'CWeatherForecastCreatorMkIIDlg' class, described above, to help create a weather sentence.

5.5.11 Weather Playlist

This class, UWeatherPlayList, is defined in 'WeatherPlayList.h' and implemented in 'WeatherPlayList.cpp'.

It has a single public member function, 'GetPlayList', which is called by the XML to HTML Translator class. It uses the function 'GetMotionFiles' implemented in the 'CWeatherForecastCreatorMkIIDlg' class, which is described above, to help create a play-list of motion files.

5.6 Notes on Functionality

Note that the user interface implemented simplifies the grammar of the Weather Forecast. The specification of the structure of a Weather Forecast has a recursive definition of some phrases. This would allow for sentences such as 'cold and dry followed by thick fog' as a Weather Condition example. The user interface implemented only allows 'cold and dry' or 'dry followed by thick fog'. A user interface that allowed recursion would be a much more complicated one to use, design and implement. Hopefully, for the purposes of the prototype this simplified user interface will suffice.

6 User Instructions

A ViSiCAST Weather Forecast consists of a single introduction sentence followed by zero or more Weather Sentences. A Weather Sentence can be one of four types:

- General – E.g. 'rain showers'

- Temperature – E.g. ‘lower temperature’
- Wind – E.g. ‘moderate wind’
- Linking – which is not an actual description of weather but denotes that any sentences following this one are for the period stated

Each type of sentence is discussed below.

6.1 Introduction Sentence

As stated above, a Weather Forecast must have a single Introduction sentence. To add, or replace an existing Introduction sentence, select the Introduction tab, choose the type of Introduction (either Full or Summary) select the appropriate text in each combo-box and then click on the Add button. The sentence created from the combo-box choices is displayed in the read-only text box just above the Add button. Note that a Summary type Introduction sentence does not require Concluding Text so that combo-box is disabled when a Summary type Introduction is selected. It is enabled if the Full type of Introduction sentence is chosen.

6.2 Linking Sentence

A Linking sentence is not a weather description sentence. It is a sentence that specifies a short period of time a few days ahead for which all following Weather Sentences apply. The description ‘Linking’ is probably not the best for this type of sentence, but at the time of writing a better one had not been suggested or found.

6.3 General, Wind and Temperature Sentences

6.3.1 Time

This is an optional part of a General, Wind or Temperature sentence. As it is optional the default value type for Time is ‘None’. Other type values are ‘Single’, ‘Or’, ‘And’ and ‘From/To’. ‘Single’ denotes a single type Time value, such as ‘in the morning’. The ‘Or’ value signifies a Time type that is a mutually exclusive choice of two values, such as ‘in the morning or in the afternoon’. The ‘And’ value type signifies the mutual inclusiveness of two values, such as ‘in the morning and in the afternoon’. The ‘From/To’ value type signifies a span of time from the first value to the second, such as ‘from in the morning to in the afternoon’.

6.3.2 Place

This is an optional part of a General, Wind or Temperature sentence. As it is optional the default value type for Place is ‘None’. Other type values are ‘Single’, ‘And’ and ‘From/To’. ‘Single’ denotes a single type Place value, such as ‘in the south’. The ‘And’ value type signifies the mutual inclusiveness of two values, such as ‘in the south and in the north’. The ‘From/To’ value type signifies an area from the first value to the second, such as ‘from in the south to in the north’.

6.3.3 General

A General Weather Sentence consists of optional Time and Place phrases and a required Weather Conditions phrase. Type values for Weather Conditions are ‘Single’, ‘And’, ‘Followed By’ and ‘Interspersed With’. The default value type for Weather Conditions is ‘Single’, an example of which is ‘rain showers’. The ‘And’ type value means that two types of Weather Conditions will occur, such as ‘rain showers and bright periods’. The ‘Followed By’ type value denotes that the first Weather Conditions will be followed by the second, such as ‘rain showers followed by bright periods’. The ‘Interspersed With’ type value denotes that two Weather Conditions are interleaved, such as ‘bright periods interspersed with rain showers’.

6.3.4 Wind

A Wind Weather Sentence consists of optional Time and Place phrases, a required Wind Force phrase and an optional Wind Direction phrase. Type values for Wind Force are ‘General’, ‘Explicit’ and ‘Explicit From/To’. The default value type for Wind Force is ‘General’, an example of which is ‘moderate wind’. The ‘Explicit’ type value denotes that the Wind Force has an actual value, such as ‘Wind-force 4’. The ‘Explicit From/To’ type value denotes that the Wind Force is in the range of the two values given, such as ‘Wind-force from 4 to 6’. The optional Wind Direction has two possible type values. These are, ‘None’, the default, and ‘Single’. An example of the ‘Single’ type value is ‘south’.

6.3.5 Temperature

A Temperature Weather Sentence consists of optional Time and Place phrases and a required Temperature phrase. Type values for Temperature are 'General' and 'Explicit'. The default value type for Temperature is 'General', an example of which is 'temperature around the freezing-point'. The 'Explicit' type value denotes that the phrase contains a value for the temperature, such as 'Temperature about 14 degrees Celsius'.

6.4 Sentences List

All sentences, except the Introduction sentence, are displayed in the Sentences list. The order in which they appear in the list defines the order that the sentences will be displayed and signed on the Weather Forecast web page. The contents of the list can be manipulated by clicking on the various buttons above it. The actions of each of these buttons are described below.

6.4.1 Add

When a Weather Sentence tab (and, therefore, not the Introduction tab) is the current tab, clicking on the Add button will cause a sentence to be built using the choices and selections of the current tab. This sentence is then added to the end of the list of sentences in the Sentences list-box.

6.4.2 Edit

This button is only enabled when a row in the Sentences list-box is selected. Clicking on this button will switch to the appropriate tab depending on the type of sentence selected, if required, and then by de-constructing the sentence will fill the tab with the choices and selections that were used to construct that sentence. The user can then amend the choices and selections to modify that sentence.

6.4.3 Update

When editing an existing sentence, as described in 6.4.2, above, this button can be clicked to update the sentence being edited. This is similar to the functionality of the Add button except that the newly constructed sentence replaces the one selected in the Sentences list, i.e. the one chosen by the user to be edited.

6.4.4 Cancel

If the Edit button has been clicked, clicking on this button cancels the editing of the selected row.

6.4.5 Delete

Clicking on this button removes the selected row from the Sentences list.

6.4.6 Up

Clicking on this button moves the selected row up one place in the Sentences list.

6.4.7 Down

Clicking on this button moves the selected row down one place in the Sentences list.

6.5 Create HTML File

Clicking on the Browse button allows the selection of a location and filename for the HTML file to be saved. The program remembers the most recently used filename, and this is displayed in the HTML file edit control. Note that the location in which the HTML file is saved should also contain the file 'visicast_logo.jpg' as this is used by the web page.

Clicking on the Create HTML File button creates an HTML file in the location specified, unless there are errors, which will be reported in a message box.

6.6 Done

Clicking on the Done button exits the program.

References

- [1] (D2-1 Document C) List of Words
- [2] (D2-1 Document B) Model for weather forecasts with mapping
- [3] (D2-1 Document 5) ViSiCAST Weather Forecast Web Page
- [4] Professional XML by Richard Anderson, et al, Wrox Press; ISBN: 1861003110
- [5] World Wide Web Consortium (W3C) XML home page: <http://www.w3.org/XML/>

ViSiCAST Milestone M5-10: Initial SiGML Definition

Project Number:	IST-1999-10500
Project Title:	ViSiCAST
	Virtual Signing: Capture, Animation, Storage and Transmission
Document Type:	Milestone Report

Milestone Number:	M5-10
Planned Date of Delivery:	June 2000
Actual Date of Delivery:	June 2000
Title of Milestone:	Initial SiGML Definition
Work-Package:	WP5 (Language and Notation)
Author(s):	JR Kennaway, JRW Glauert, R Elliott, and K Parsons (UEA)

Abstract:

This internal milestone marks the successful completion of the first stage of definition of the SiGML notation.

[By agreement within the project, "SiGML" is the new name for the notation originally designated "GML" in the project proposal.]

The definition consists of three (text) files, of which the second and third are auxiliary definitions, supplementing the first:

- (a) sigml_200000728.dtd;
- (b) gml_20000803.dtd;
- (c) hamgram_annot_20000803.edt.

The first of these (a) is the DTD (document type definition, or XML grammar) defining SiGML itself: this defines a language approximately equivalent in expressive power, and broadly similar in structure to, HamNoSys. It is a streamlined version of the DTD (b), which is essentially a reworking of the HamNoSys grammar in DTD notation. The HamNoSys grammar itself, annotated with informal semantic definitions, is given in (c). As yet there is no separate semantic definition for SiGML, but the semantics should be clear to anyone familiar with HamNoSys, and hence to anyone who follows the auxiliary definitions (b) and (c).

A prototype tool is also available for converting HamNoSys into SiGML.

ViSiCAST “HamNoSys-to-SiGML Translation”

R Elliott, JR Kennaway, K Parsons
SYS, UEA Norwich
{re,jrk,kjp}@sys.uea.ac.uk

September, 2000

1 Description of Work in Progress

We give here an outline of work currently in progress on translation from *HamNoSys* to *SiGML*. This is currently performed in two stages:

- translation from *HamNoSys* to “old” *SiGML* (sometimes referred to as plain *GML*);
- translation from “old” *SiGML* to “new” *SiGML* (the latter of which is sometimes referred to as “revised” *SiGML*).

1.1 From *HamNoSys* to “Old” *SiGML*

The first of the two stages is performed by a Java program, itself generated using Terence Parr’s *Antlr* translator generation system (available from www.antlr.org). The relevant source files and build scripts are held in the main `hamToSiGML` folder. This folder also contains a text file, `hamgram_annot-20000803.{txt,edt}`, defining the input *HamNoSys* grammar annotated with informal definitions of the intended semantics. The Java bytecode for the translator is held in the `hpclasses.jar` file. The translator can be run using a command of the form:

```
genSigml hamnosys-input-file sigml-output-file
```

The result of applying the translator to the *HamNoSys* sign for “schmaltz” are shown in the *SiGML* output file `test.gml`. The file `test.out` shows the corresponding AST (abstract syntax tree), whose structure, it may be observed, is very close to that of the *SiGML* output — reflecting the (intentionally) very close similarity of structure between *HamNoSys* and “old” *SiGML*.

1.2 From “Old” *SiGML* to “New” *SiGML*

For the second translation stage, a different approach has been adopted: as both the input to, and the output from, this stage are XML, it has been implemented as an application of the XSLT transformation notation.

The relevant files are held in the `sigmlTransform` sub-folder of the main folder:

- The DTD’s for “old” and “new” *SiGML* are in the respective files `gml-20000728.dtd` and `sigml_20000728.dtd`.
- The definition of the XSL transformation between these two is held in `sigml.xsl`.
- `gml100July27.xml` is a simple “old” *SiGML* test file.
- `sigml100July27.xml` is the corresponding “new” *SiGML* file, generated using the given XSL definition (see below).
- Scripts to check the validity of these input and output files are held in the `check{Old,New}SigmlValid.vbs` (VBScript) files.

The XSL translation generating the “new” *SiGML* file corresponding to the given test file can be performed using James Clark’s *xt* translator (www.jclark.com), or using Microsoft’s *MSXML* library (msdn.microsoft.com/xml).

Again, testing to date has been very limited, due to the limited number of examples as yet available via the first stage.

ViSiCAST Milestone M2-1: SiGML Tool Initial

Project Number:	IST-1999-10500
Project Title:	ViSiCAST
	Virtual Signing: Capture, Animation, Storage and Transmission
Document Type:	Milestone Report

Milestone Number:	M2-1
Planned Date of Delivery:	July 2000
Actual Date of Delivery:	July 2000
Title of Milestone:	SiGML Tool Initial
Work-Package:	WP2 (MM and WWW Applications)
Author(s):	R Elliott (UEA)

Abstract:

This internal milestone marks the successful development of a Java application which demonstrates the ability to drive the Televirtual signing avatar (IHost) with signs taken from a streamed sequence expressed in prototype SiGML.

For the purposes of this demonstration, the SiGML definition has been enhanced to allow a sign to be characterised by a “gloss level” definition, containing essentially the same parameters as those used to drive the IHost signing avatar. The application associated with this milestone performs three main functions:

- (a) acts as a “**queue server**” (running in its own thread), providing sign parameters to the signing avatar;
- (b) acts as a **parser** (again in its own thread) for the input stream of SiGML data, extracting the sign parameters in real-time;
- (c) provides a **buffer** via which the sign parameters generated by (b) are fed (in real-time) to the server (a); for demonstration purposes, the contents of this buffer are displayed dynamically.

For further details, refer to the description document (README) accompanying the software.

ViSiCAST “Streaming Prototype SiGML Demonstration”

R Elliott, SYS, UEA Norwich
re@sys.uea.ac.uk

September, 2000

1 Description

We describe here a Java application, developed in July, which demonstrates the ability to drive the *SignAnim* 3.2 avatar (*IHost*) with signs taken from a streamed sequence expressed in prototype *SiGML*.

This application runs several threads, of which the two most important are:

- a thread acting as avatar sign queue server;
- a thread which reads and parses a stream of *SiGML* signs.

These two threads stand in a producer-consumer relationship to each other (with the queue server in the role of consumer), communicating by means of a buffer of *Lexicon Sign Data* items, each of which specifies a sign name and an associated playing speed value. For demonstration purposes this buffer has an associated real-time status display panel. This display shows:

- the stream index of the buffer’s *Base* entry (i.e. the current input stream index for the consumer);
- the stream index of the buffer’s *Limit* entry (i.e. the current input stream index for the producer);
- the buffer *Size*, its current number of entries (i.e. the difference between the previous two values).

To make the streaming demonstration accessible to the human viewer, the parsing of the *SiGML* sign stream is slowed down artificially, effectively simulating the delivery of the sequence, which the demonstrator actually reads from a local text file, over an (implausibly slow) network connection. The extent of this degradation can be controlled by adjusting the value of the static variable `lsDelay` in the main `QServeApp` class.

For the purposes of this demonstration, the draft *SiGML* DTD has been augmented to allow each `<gmlsign>` element to include an optional *lexicon sign*

(<lexsign>) element, representing a *SignAnim* 3.2 avatar sign queue entry, as defined by the following fragment from the DTD file, `re_rgm100Jun08.dtd`:

```
<!ELEMENT gmlsign (lexsign?, sign?, motiondata?)>

<!ELEMENT lexsign (mocapdata, mocapspeed)>

<!ELEMENT mocapdata EMPTY>
<!ATTLIST mocapdata
    filename    CDATA    #REQUIRED >

<!ELEMENT mocapspeed EMPTY>
<!ATTLIST mocapspeed
    percent    CDATA    #REQUIRED >
```

For the demonstration to work, <lexsign> elements must refer to signs for which *SignAnim* 3.2 motion capture data is available. Hence, in the corresponding prototype *SiGML* file, `re_rgm100Jun08.xml`, the *SiGML* definitions for authentic DGS signs such as ARTEN (for which motion capture data is *not* currently available) have had inappropriate lexicon sign names such as `duck` (for which motion capture data *is* available) grafted on to them.

2 Running the Demonstration

To run this demonstration, the following are required:

- Sun's Java implementation – JDK 1.2.2 (or later); (to run only, without compiling, JRE is sufficient, instead of the full JDK);
- Sun's XML Parsing classes – JAXP 1.0 (or later);
- Televirtual's IHost Avatar, and associated Motion Capture files, or the simulator `avatarsim.tcl` – all of which are included in *SignAnim* 3.2.

To compile and run the demonstrator program, run the batch script `qsapp-buildrun.bat`. This produces the demonstrator Java bytecode in the file `qs-classes.jar`.

To run the previously compiled demonstrator program, run the script `qs-run.bat`.

Once the demonstrator has been launched, the avatar (or avatar simulator) should be launched as a separate process¹, and, in the case of the full avatar, the connection should then be established with the queue server (on the standard *SignAnim* socket 9001).

To start the parsing of the *SiGML* sign sequence, press the *Start* button on the demonstrator application's Buffer Display panel: completion of the parse

¹Given the appropriate configuration parameters, this process may be run on a different host system to that running the streamed sign server.

is indicated when the displayed *Limit* value reaches the number of signs in the stream (20, for the current input file), and also by a message (“GML Document input completed”) on the application’s console. The streaming capabilities of the processor are demonstrated by the fact that successive signs appear in the display, and are then fed to the avatar if it is present and properly connected, *prior* to completion of parsing.